SIEMENS

COB1 (BS2000)
COBOL Compiler

Reference Manual
Part 1

Edition August 1986 (Software Product COB1 V2.3A)

Order No. U343-J-Z55-6-7600
Printed in the Federal Republic of Germany
3810 AG 11860.8 (4770)

Order No.:
U343-J-Z55-6-7600

A ring binder for this manual is available at a cost of DM 4,20
Order number: U1225-J-Z18-1

SIEMENS

COB1 (BS2000)
COBOL Compiler

Reference Manual
Part 1

Edition August 1986 (Software Product COB1 V2.3A)

	Order No. U3413-J-Z55-8-7600	A ring binder for this manual is available at a cost of DM 4,20 Order number: U7259-J-Z1-1

COB1 (BS2000) COBOL Compiler

Reference Manual Part 1

Edition August 1986 (Software Product COB1 V2.3A)

Order No. U343-J-Z55-6-7600
Printed in the Federal Republic of Germany
3810 AG 11860.8 (4770)

This document shall not be duplicated, nor its contents used for any purpose, unless express permission has been granted.

Performance features may be added, modified or omitted in the course of product development. Other information in this publication is subject to the same conditions.

Siemens Aktiengesellschaft

PREFACE

This manual describes the COBOL language for the Siemens COBOL compiler COB1, Version 2.3A, BS2000 Operating System.

The manual contains all of those language elements which might possibly occur in the creation of COBOL programs, divided according to function, format, rules, programming considerations and examples.

It is intended as a guide to the writing of COBOL programs and as a complement to training manuals. Knowledge of COBOL is a prerequisite.

The structure follows the language modules defined in the Standard for COBOL. Extensions to and deviations from Standard 74 are marked with colored shading as follows:

Gray shading = Siemens COB1 extensions
Orange shading = Language elements to be avoided wherever possible since they will no longer be supported by future COBOL standards.

The concepts and expressions used in this manual are defined, in alphabetical order, in the "Glossary".

For further prefatory information consult chapter 1, "Introduction".

Changes made to the previous manual are summarized in the "List of Amendments".

Literature references in the text are given in short-title format. The complete title of each publication cited is listed in the "Literature" section.

To help us continue to improve our publications, please send us your comments, requests and suggestions using the pink reply forms provided.

Editorial Department K D ST QM 2

ACKNOWLEDGMENT

COBOL, the programming language described in this manual, is based on the language laid down in the standard document "American National Standard Programming Language COBOL, X.3.23-1974". In recognition of the efforts made to develop and standardize COBOL it is customary to precede a description of COBOL with the following text as it appears in the original publication:

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention COBOL in acknowledgment of the source, but need not quote the acknowledgment.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

LIST OF AMENDMENTS

**The following amendments have been made since publication of the
Corrections February 1986 (COB1 Version 2.2A)
and incorporated in the
Edition August 1986 (COB1 Version 2.3A)**

1. All references to BS1000 have been removed.

2. The following language elements are no longer described since they will not be supported by future versions of the compiler:

Statements	ALTER
	CALL USING procedure-name
	*DEBUG
	ENTER
	EXAMINE
	EXHIBIT
	GO TO to/from USE procedures
	ON
	TRACE
	TRANSFORM
Phrases	ALTERNATE in RESERVE clause
	COBRUN QUOTE1
	COMP-4
	CURRENT DATE/TODAYS DATE
	FOR MULTIPLE REEL
	OTHERWISE
	POSITIONING
Clauses	FILE-LIMIT
	PROCESSING MODE
	SYMBOLIC KEY/NOMINAL KEY
	TRACK-AREA
	VALUE OF
Paragraphs	DATE-COMPILED
	REMARKS

3. New language elements:

   FILE STATUS...data-name-2
   INITIALIZE
   START KEY LESS
   SYSLSTnn support

4. Other important changes:

Page	Item changed	new	modi-fied	dis-carded
1-6	Character set: lower case	X		
1-41	RETURN-CODE: gray shading	X		
1-50	Numeric floating-point literals: format		X	
2-2	Headings of program divisions		X	
2-15	SPECIAL-NAMES: format		X	
2-32	77-level description entry		X	
2-42	Data description entries for group items		X	
2-65	Example 22		X	
2-91	COMPUTATIONAL: Rule e)			X
2-91	Length of paragraph/section names		X	
2-100	Relation Condition: table		X	
2-103	Note 4			X
2-125	DAY-OF-WEEK: gray shading			X
2-127	ADD statement: format 2		X	
2-133	DISPLAY: Programming consideration 6	X		
2-142	GO TO statement, format 3: orange shading	X		
3-12	ORGANIZATION clause: format		X	
3-20	File description entry: rules		X	
3-32	RECORD clause: format 1 + rules		X	
3-35	RECORDING MODE: rules		X	
3-46	READ statement: rule 5		X	
4-15	ORGANIZATION clause: format		X	
4-23	File description entry: rules		X	
5-6	Message 94		X	
5-14	ORGANIZATION clause: format		X	
5-22	File description entry: rules		X	
7-6	CALL statement: rule 4		X	
7-11	EXIT PROGRAM: rules		X	
8-10	Control fields: rule 8			X
	Literature		X	

CONTENTS

		Page
1	**General Concepts**	**1-1**
1.1	Introduction	1-1
1.2	Glossary	1-3
1.3	Notation	1-30
1.4	Language Concepts	1-33
1.4.1	COBOL Character Set	1-33
1.4.2	Language Structure	1-34
1.4.2.1	Separators	1-34
1.4.2.2	Character-Strings	1-36
1.4.2.3	COBOL Words	1-36
1.4.2.4	Concept of Computer-Independent Data Description	1-51
1.4.2.5	Implementor-Dependent Representation and Alignment of Data	1-63
1.4.2.6	Qualification	1-69
1.5	Declaratives	1-71
1.6	Statements and Sentences	1-72
1.6.1	Conditional Statements and Conditional Sentences	1-72
1.6.2	Compiler-Directing Statements and Compiler-Directing Sentences	1-72
1.6.3	Imperative Statements and Imperative Sentences	1-73
1.6.4	Categories of Statements	1-74
1.7	Program Image Format	1-76
1.7.1	General Description	1-76
1.7.2	Rules for Using the COBOL Coding Form	1-78
1.8	Creating a COBOL Program	1-82
1.9	Processing a COBOL Program	1-83
1.10	EBCDIC Character Set	1-84
2	**Basic Elements of a COBOL Program**	**2-1**
2.1	Introduction	2-1
2.2	Identification Division	2-4
2.2.1	General Description	2-4
2.2.2	Structure	2-4
2.2.3	Paragraphs	2-5
	PROGRAM-ID Paragraph	2-5
	DATE-COMPILED Paragraph	2-6
2.3	Environment Division	2-7
2.3.1	General Description	2-7
2.3.2	Structure	2-10
2.3.3	Configuration Section	2-11
	SOURCE-COMPUTER Paragraph	2-12
	OBJECT-COMPUTER Paragraph	2-13
	SPECIAL-NAMES Paragraph	2-15
2.3.4	INPUT-OUTPUT Section	2-25
2.4	Data Division	2-26
2.4.1	General Description	2-26
2.4.2	Structure	2-27
2.4.2.1	General Rules	2-29
2.4.2.2	FILE SECTION	2-30
2.4.2.3	WORKING-STORAGE SECTION	2-31
2.4.2.4	LINKAGE SECTION	2-32
2.4.2.5	REPORT SECTION	2-33
2.4.3	Organization of the Entries of the Data Division	2-34
2.4.4	77-Level Entries in the WORKING-STORAGE and LINKAGE Sections	2-36

2.4.5	Data Description Entry	2-37
2.4.5.1	General Description	2-37
2.4.5.2	Level Numbers	2-39
2.4.5.3	Data Description Entries for Group Items within a Record Description in the FILE, WORKING-STORAGE and LINKAGE Sections	2-41
2.4.5.4	Data Description Entries for Elementary Items within a Record Description in the FILE, WORKING-STORAGE and LINKAGE Sections	2-42
2.4.5.5	Clauses for Data Description	2-43
	BLANK WHEN ZERO Clause	2-43
	Data-Name or FILLER Clause	2-44
	JUSTIFIED Clause	2-45
	OCCURS Clause	2-46
	PICTURE Clause	2-47
	REDEFINES Clause	2-63
	RENAMES Clause	2-66
	SIGN Clause	2-69
	SYNCHRONIZED Clause	2-73
	USAGE Clause	2-76
	VALUE Clause	2-84
2.5	Procedure Division	2-89
2.5.1	General Description	2-89
2.5.2	Structure	2-90
	General Format	2-90
2.5.3	Arithmetic Expressions	2-92
2.5.3.1	Arithmetic Operators	2-92
2.5.3.2	Rules for the Formation and Evaluation of Expressions	2-93
2.5.4	Conditions	2-95
2.5.4.1	Simple Conditions	2-96
	Condition-Name Condition	2-96
	Class Condition	2-97
	Switch-Status Condition	2-99
	Relation Condition	2-100
	Sign Condition	2-104
2.5.4.2	Compound Conditions	2-105
2.5.4.3	Implied Subjects and Relational Operators	2-108
2.5.5	Arithmetic Statements	2-110
2.5.6	Options in Arithmetic Statements	2-113
	CORRESPONDING Phrase	2-113
	GIVING Phrase	2-116
	ROUNDED Phrase	2-117
	SIZE ERROR Phrase	2-118
2.5.7	Overlapping Operands	2-120
2.5.8	Incompatible Data	2-121
2.5.9	Statements	2-122
	ACCEPT Statement	2-122
	ADD Statement	2-126
	ALTER Statement 1)	2-129
	COMPUTE Statement	2-131
	DISPLAY Statement	2-132
	DIVIDE Statement	2-135
	EXIT Statement	2-139
	GO TO Statement	2-141
	IF Statement	2-144
	INITIALIZE Statement	2-147
	INSPECT Statement	2-149
	MOVE Statement	2-157
	MULTIPLY Statement	2-164
	PERFORM Statement	2-166
	SET Statement	2-185
	STOP Statement	2-186
	STRING Statement	2-187

	SUBTRACT Statement	2-190
	UNSTRING Statement	2-193
3	**Sequential File Organization**	**3-1**
3.1	File Concepts	3-1
3.1.1	Sequential Organization	3-1
3.1.2	Sequential Access to Data Records of a Sequential File	3-1
3.1.3	I-O Status	3-2
3.2	Language Elements of the Environment Division	3-5
3.2.1	INPUT-OUTPUT SECTION	3-5
3.2.1.1	FILE-CONTROL Paragraph	3-6
	SELECT Clause	3-7
	ASSIGN Clause	3-8
	ACCESS MODE Clause	3-10
	FILE STATUS Clause	3-11
	ORGANIZATION Clause	3-12
	RESERVE Clause	3-13
3.2.1.2	I-O-CONTROL Paragraph	3-14
	MULTIPLE FILE TAPE Clause	3-15
	RERUN Clause	3-16
	SAME AREA Clause	3-18
3.3	Language Elements of the Data Division	3-19
3.3.1	FILE SECTION	3-19
3.3.1.1	File Description (FD) Entry	3-20
	BLOCK CONTAINS Clause	3-22
	CODE-SET Clause	3-24
	DATA RECORDS Clause	3-25
	LABEL RECORDS Clause	3-26
	LINAGE Clause	3-29
	RECORD Clause	3-32
	RECORDING MODE Clause	3-35
3.4	Language Elements of the Procedure Division	3-37
3.4.1	Input/Output Statements	3-37
	CLOSE Statement	3-38
	OPEN Statement	3-42
	READ Statement	3-46
	REWRITE Statement	3-49
	USE Statement	3-51
	WRITE Statement	3-60

Contents

4	**Relative File Organization**	**4-1**
4.1	File Concepts	4-1
4.1.1	Relative Organization	4-1
4.1.2	Sequential Access to Data Records in a Relative File	4-1
4.1.3	Random Access to Data Records of a Relative File	4-2
4.1.4	Dynamic Access to Data Records in a Relative File	4-3
4.1.5	Valid Access Methods	4-3
4.1.6	I-O Status	4-4
4.2	Language Elements of the Environment Division	4-8
4.2.1	INPUT-OUTPUT SECTION	4-8
4.2.1.1	FILE-CONTROL Paragraph	4-9
	SELECT Clause	4-10
	ASSIGN Clause	4-11
	ACCESS MODE Clause	4-13
	FILE STATUS Clause	4-14
	ORGANIZATION Clause	4-15
	RELATIVE KEY Clause	4-16
	RESERVE-Clause	4-17
4.2.1.2	I-O-CONTROL-Paragraph	4-18
	RERUN-Clause	4-19
	SAME AREA Clause	4-21
4.3	Language Elements of the Data Division	4-22
4.3.1	FILE SECTION	4-22
4.3.1.1	File Description (FD) Entry	4-23
	BLOCKS CONTAINS Clause	4-25
	DATA RECORDS Clause	4-26
	LABEL RECORDS Clause	4-27
	RECORD Clause	4-28
	RECORDING MODE Clause	4-29
4.4	Language Elements of the Procedure Division	4-30
4.4.1	INVALID KEY Condition	4-30
4.4.2	Input-Output Statements	4-31
	CLOSE Statement	4-32
	DELETE Statement	4-34
	OPEN Statement	4-35
	READ Statement	4-37
	REWRITE Statement	4-41
	START Statement	4-43
	USE Statement	4-45
	WRITE Statement	4-49

5	**Indexed File Organization**	**5-1**
5.1	File Concepts	5-1
5.1.1	Indexed Organization	5-1
5.1.2	Sequential Access to Data Records of an Indexed File	5-1
5.1.3	Random Access to Data Records of an Indexed File	5-2
5.1.4	Dynamic/Extended Access to Data Records of an Indexed File	5-2
5.1.5	Valid Access Methods	5-2
5.1.6	I-O Status	5-3
5.2	Language Elements of the Environment Division	5-7
5.2.1	INPUT-OUTPUT SECTION	5-7
5.2.1.1	FILE-CONTROL Paragraph	5-8
	SELECT Clause	5-9
	ASSIGN Clause	5-10
	ACCESS MODE Clause	5-12
	FILE STATUS Clause	5-13
	ORGANIZATION Clause	5-14
	RECORD KEY Clause	5-15
	RESERVE Clause	5-17
5.2.1.2	I-O-CONTROL Paragraph	5-18
	RERUN Clause	5-19
	SAME AREA Clause	5-21
5.3	Language Elements of the Data Division	5-22
5.3.1	FILE SECTION	5-22
5.3.1.1	File Description (FD) Entry	5-23
	BLOCK CONTAINS Clause	5-25
	DATA RECORDS Clause	5-27
	LABEL RECORDS Clause	5-28
	RECORD Clause	5-29
	RECORDING MODE Clause	5-32
5.4	Language Elements of the Procedure Division	5-34
5.4.1	INVALID KEY Condition	5-34
5.4.2	Input/Output Statements	5-35
	CLOSE Statement	5-36
	DELETE Statement	5-38
	OPEN Statement	5-39
	READ Statement	5-41
	REWRITE Statement	5-46
	START Statement	5-48
	USE Statement	5-50
	WRITE Statement	5-54

6	**Table Handling**	**6-1**
6.1	General Description	6-1
6.2	Table Definition	6-2
6.2.1	One-Dimensional Tables	6-2
6.2.2	Multi-Dimensional Tables	6-3
6.2.3	Initial Values of Table Items	6-4
6.2.4	References to Table Items	6-5
6.3	Subscripting	6-6
6.3.1	Direct Subscripting	6-7
6.3.2	Relative Subscripting	6-7
6.3.3	Formats	6-8
6.4	Indexing	6-10
6.4.1	Direct Indexing	6-10
6.4.2	Relative Indexing	6-11
6.4.3	Formats	6-11
6.5	Indexing and Subscripting Compared	6-14
6.5.1	Availability of Occurrence Numbers for the User	6-14
6.5.2	References to Table Items	6-14
6.5.3	Changing the Index	6-14
6.6	Language Elements	6-15
6.6.1	Language Elements of the Data Division	6-16
	OCCURS Clause	6-16
	USAGE Clause	6-24
6.6.2	Language Elements of the Procedure Division	6-26
	SEARCH Statement	6-26
	SET Statement	6-35
	Comparisons Involving Index-Names and/or Index Data Items	6-40
7	**Program Communication**	**7-1**
7.1	General Description	7-1
7.2	Language Elements	7-2
7.2.1	Data Division Language Elements	7-3
	LINKAGE SECTION	7-3
7.2.2	Procedure Division Language Elements	7-4
	Procedure Division Header	7-4
	CALL Statement	7-6
	ENTRY Statement	7-9
	EXIT PROGRAM Statement	7-11

8	**Report Writer**	**8-1**
8.1	General Description	8-1
8.2	General Description of the Data Division	8-2
8.3	General Description of the Procedure Division	8-4
8.4	Data Division Language Elements	8-5
8.4.1	REPORT Clause	8-5
8.4.2	REPORT SECTION	8-6
8.4.2.1	Report Description Entries	8-7
	CODE Clause	8-8
	CONTROL Clause	8-9
	PAGE LIMIT Clause	8-12
8.4.2.2	Report Group Entries	8-18
	COLUMN Clause	8-22
	GROUP INDICATE Clause	8-24
	LINE Clause	8-26
	NEXT GROUP Clause	8-31
	SOURCE Clause	8-34
	SUM Clause	8-36
	TYPE Clause	8-46
8.5	Language Elements of the Procedure Division	8-50
	GENERATE Statement	8-50
	INITIATE Statement	8-52
	TERMINATE Statement	8-53
	USE BEFORE REPORTING Statement	8-54
8.6	Special Registers of the Report Writer	8-56
	LINE-COUNTER Special Register	8-56
	PAGE-COUNTER Special Register	8-57
	PRINT-SWITCH Special Register	8-58
	CBL-CTR Special Register	8-59
9	**Segmentation Feature**	**9-1**
9.1	General Description	9-1
9.2	Organization	9-1
9.3	Fixed Portion of the Object Program	9-1
9.4	Independent Segments	9-2
9.5	General Rules for Segmentation	9-2
9.6	Language Elements	9-5
9.6.1	Language Elements of the Environment Division	9-6
	SEGMENT-LIMIT Clause	9-6
9.6.2	Language Elements of the Procedure Division	9-7
	Segment Number	9-7

Contents

10	**Sort-Merge Feature**	**10-1**
10.1	Sorting and Merging of Files	10-1
10.1.1	Sort Processing	10-1
10.1.2	Merge Processing	10-2
10.1.3	Sort and Merge without Input/Output Procedures	10-2
10.1.4	Sort with Input/Output Procedures of the User	10-3
10.1.5	Sort and Merge with Output Procedures but without Input Procedure of the User	10-3
10.2	Language Elements	10-4
10.2.1	Language Elements of the Environment Division	10-6
	RERUN Clause	10-6
	SAME AREA Clause	10-7
	SELECT Clause	10-8
10.2.2	Language Elements of the Data Division	10-10
	Sort-File Description	10-10
10.2.3	Language Elements of the Procedure Division	10-12
	MERGE Statement	10-12
	RELEASE Statement	10-17
	RETURN Statement	10-18
	SORT Statement	10-20
10.3	Special Registers for SORT	10-26
	Examples	10-28
11	**Debugging Aids**	**11-1**
11.1	General Description	11-1
11.2	Environment Division Language Elements	11-2
	Debugging Line	11-2
	WITH DEBUGGING MODE Clause	11-3
12	**Libraries**	**12-1**
12.1	COPY Statement	12-1

Literature

TABLES

TABLES AND FIGURES IN CHAPTER 1

Tables

1-1 COBOL Character Set 1-33

1-2 Punctuation Marks, Arithmetic Operators, and Relational Operators 1-34

1-3 COBOL User-Defined Words 1-38

1-4 Types of COBOL Connectives 1-40

1-5 COBOL Special Registers 1-41

1-6 COBOL Figurative Constants and Values 1-45

1-7 Reserved COBOL Words 1-46

1-8 Classes and Categories of Elementary and Group Items 1-57

1-9 COBOL Margin Conventions 1-80

Figures

1-1 Relationship between Group Items and Elementary Items in a Data Record 1-55

1-2 Group Items and Elementary Items in a Data Record 1-55

1-3 COBOL Coding Form 1-77

TABLES AND FEATURES IN CHAPTER 2

Tables

2-1 General Structure of a COBOL Source Program 2-2

2-2 Valid Access Methods for the Different Types of File Organization 2-9

2-3 Meaning of Implementor Names 2-16

2-4 Printer Paper Feed Using Implementor Names 2-16

2-5 Summary of Level Indicators 2-34

2-6 Summary of Level Numbers 2-35

2-7 Precedence of Symbols used in the PICTURE Clause 2-48

2-8 Editing Sign Control Symbols and their Results 2-59

2-9 Internal Representation of Number Data Items 2-83

2-10 Valid Symbol Combinations in Arithmetic Expressions 2-93

2-11 Valid Forms of the Class Test 2-98

2-12 Relational Operators 2-100

2-13 Valid Comparisons between Various Operands 2-103

2-14 Logical Operators 2-105

2-15 Valid Symbol Pairs of Conditions and Logical Operators 2-106

2-16 Use of Logical Operators 2-107

2-17 Calculating the Integer and Decimal Places in Intermediate Results 2-111

2-18 Maximum Logical Record Size for the DISPLAY Statement 2-133

2-19 Categories of Elementary Items 2-160

2-20 Permissible Elementary and Group Moves 2-161

2-21 Rules for Alphanumeric Moves 2-162

2-22 Rules for Numeric Moves 2-163

Figures

2-1 Conditional Statements with Nested IF Statements 2-145

TABLES IN CHAPTER 3

3-1 Device Names 3- 9

3-2 Functions of File Description Clauses 3-21

3-3 Calculation of the Maximum Block Size 3-23

3-4 LABEL RECORDS Clause Consideration 3-28

3-5 Setup of a Logical Page 3-30

3-6 Relationship between the RECORD Clause and the RECORDING MODE Clause 3-36

3-7 Relation of Types of Sequential Files and the CLOSE Statement Options 3-41

3-8 Valid Input-Output Statements in each OPEN Mode 3-45

3-9 Format and Meaning of Spacing Indicators in a WRITE ADVANCING Statement for PRINTER and PRINTERHOOD 3-66

3-10 Use of Printer Device Name in Connection with the WRITE Statement 3-67

TABLES IN CHAPTER 4

4-1 Valid Access Methods 4- 3

4-2 Functions of File Description Clauses 4-24

4-3 Relationship of Types of Random-Access Files and CLOSE Statement Options 4-32

4-4 Valid Input-Output Statements in each OPEN Mode 4-36

4-5 WRITE Statement. Causes of INVALID KEY Conditions 4-51

TABLES IN CHAPTER 5

5-1 Valid Access Methods 5- 2

5-2 Functions of File Description Clauses 5-24

5-3 Calculation of the Maximum Block Size 5-26

5-4 Relationship between the RECORD Clause and the RECORDING MODE Clause 5-33

5-5 Relationship of Types of Random-Access Files and CLOSE Statement Options 5-36

5-6 Valid Input Output Statements in each OPEN Mode 5-40

5-7 WRITE Statement Causes of INVALID KEY Conditions 5-56

TABLES IN CHAPTER 6

Tables

6-1 Valid uses of the SET Statement 6-36

6-2 Index-Names and Index Data Items - Valid Comparisons 6-40

Figures

6-1 Schematic Representation of TABLE 6- 3

6-2 Format-1 SEARCH statement Containing two WHEN Conditions 6-29

TABLES IN CHAPTER 8

8-1 Report Group Types Used in Report Group Description..... 8- 3

8-2 Page Partitioning into Regions (Schematic) 8-15

8-3 Report Group Entry Clauses 8-21

8-4 CBL-CTR Values in USE Procedures for Page Headings and Control Headings 8-61

8-5 CBL Values in USE Procedures for Page Footings and Control Footings 8-61

General Concepts

Basic Elements of a COBOL Program

Sequential File Organization

Part 1

Order No.:

U343-J-Z55-6-7600

Relative File Organization

Indexed File Organization

Table Handling

Program Communication

Report Writer

Segmentation Feature

Sort-Merge Feature

Debugging Aids

Part 2

Order No.:

U996-J-Z55-6-7600

Libraries

1 GENERAL CONCEPTS

1.1 INTRODUCTION

In most cases, the solving of commercial problems involves processing large amounts of data. COBOL is particularly well suited to this task. COBOL programs are largely independent of the peculiarities of individual hardware systems. The language is laid down clearly and precisely in an official document issued by the American National Standard Institute (ANSI) under the title

American National Standard
Programming Language COBOL
ANSI X3.23-1974

this being a revised version of the earlier edition of 1968.

The German standard version DIN 66028 and the international standard version ISO 1989-1978 correspond to American National Standard.

Wherever the term "Standard COBOL" appears in this volume it refers to the COBOL language set as laid down by ANSI in 1974, or by ISO and DIN.

For the purpose of description, the ANSI publication divides COBOL into a nucleus and 11 functional modules such as Table Handling, Sequential File Organization etc. In turn, each of these modules contains two or three function configuration levels, with each of the lower levels representing a genuine subset of the higher levels of the same module.

The COBOL compiler described here, COB1, largely corresponds to level 2 of Standard 74 as regards its language set. The Data Communication language module is not supported.

A number of Siemens-specific extensions have been dropped from the manual since they will no longer be supported by future versions of the compiler. These "obsolete" language elements are listed in the "List of Amendments". The Siemens-specific language extensions that remain valid are printed on gray shading. Language elements no longer supported by future COBOL standards are printed on orange shading.

This description is intended to support the COBOL programmer when writing his programs. It is neither a training manual nor a user's guide. The "COB1 User's Guide" [7] describes how to use the compiler.

The descriptions of the language elements contain separate sections on function, format, rules and programming

The **function** section offers a concise, general description of the individual language elements. If several formats are involved, the functional differences between them are explained in brief.

The **format** section defines the specific arrangement of character strings and separators required for a valid clause, statement, or compound structure. The occurrence of specific strings and separators and their order of appearance as shown in the format section are decisive.

The specific notation used for describing the formats is explained under the heading "General Format".

Where more than one specific statement is permitted, the variant formats are designated as "**format-1**", "**format-2**" and so forth.

The **rules** section describes the particular requirements and restrictions for a given function and offers additional explanations and application guidelines.

Programming considerations describe the use of the language structure within the program context, that is, as a function of previous and subsequent as well as superior and subordinate structures and in conjunction with references and cross-references from other language elements which, strictly speaking, are independent of the described structure. Restrictions on the order of effects at program runtime are discussed. Generally speaking, these considerations are concerned with those elements which do not appear directly in the format section.

1.2 GLOSSARY

Introduction

The expressions defined below are used in this manual for describing the COBOL language. The meanings of these expressions for COBOL do not necessarily apply to other programming languages.

The definitions are brief summaries of basic characteristics. For detailed explanations and syntax rules consult the later chapters of this volume.

Access Mode

The manner in which records are to be operated upon within a file.

Actual Decimal Point

The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Actual Key

A key used to express the physical location of a logical record in a file. The first part of this key is the track indicator, the second part the record indicator.

Alphabetical Character

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
plus the space.

Alphabet-Name

A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

Alphanumeric Character

Any character in the computer's character set.

Area A

Columns 8 thru 11 on the COBOL coding sheet.

Area B

Columns 12 thru 72 on the COBOL coding sheet.

Arithmetic Expression

An arithmetic expression can be:

- an identifier or a numeric elementary item
- a numeric literal
- such identifiers and literals separated by arithmetic operators
- two arithmetic expressions separated by an arithmetic operator
- an arithmetic expression enclosed in parentheses.

Arithmetic Operator

A single character, or a fixed two-character combination, that belongs to the following set:

Character	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Ascending Key

A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point

A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

AT END Condition

An AT END condition may occur:

1. During execution of a sequential READ statement for a file.
2. During execution of a RETURN statement whenever there is no logical record for the sort or merge file.
3. During execution of a SEARCH statement whenever the search terminates before any of the WHEN conditions have been satisfied.

Binary Search

A method of searching a table in ascending or descending order for a particular element. The search takes place by a process of halving the searched area. At each stage of the search, the middle element is compared to see whether it is greater than, less than or equal to the element being sought. This process of halving and comparing continues until the checked element identical to the element being sought.

Blank Lines

Lines filled entirely with blanks in columns 7 thru 72 on the COBOL coding sheet.

Block

A physical unit of data that is normally composed of one or more logical records or a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

Body Group

Generic name for a report group, control heading or control footing.

Called Program

A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

Calling Program

A program which executes a CALL to another program.

Character

The basic indivisible unit of the language.

Character-String

A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

Class Condition

The class condition establishes whether the contents of a data item are
completely numeric,
completely alphabetic,
completely uppercase,
completely lowercase, or
completely made up of characters defined by means of the class-name specified in the SPECIAL-NAMES paragraph of the Environment Division.

Class-Name

A user-defined word specified in the SPECIAL-NAMES paragraph of the Environment Division and naming a character set defined by the user. The Class-Name is entered in the class condition for purposes of checking whether a data items consists entirely of characters from this character set.

Clause

A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Character Set

The complete COBOL character set listed below consists of 51 characters, including the 26 uppercase letters, 10 digits, the space, and 14 special characters:

COBOL Character Set

The complete COBOL character set listed below consists of 51 characters, including the 26 uppercase letters, 10 digits, the space, and 14 special characters:

Character	Meaning
0 to 9	Digit
A to Z, a to z	Letter
	Space (blank)
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol

COBOL Word

(See Word)

Collating Sequence

The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging, and comparing.

Column

A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

Combined Condition

A condition that is the result of connecting two or more conditions with the "AND" or the "OR" logical operator.

Comment Entry

An explanatory entry in the Identification or Procedure Division of a source program.

Comment line

A source program line containing an asterisk (*) or slash (/) in the indicator area of the line, i.e. in column 7 of the COBOL coding form.

Any combination of characters from the computer's character set may appear in areas A and B. The comment line serves only for documentation in a program.

The asterisk indicates a comment line. A slash indicates a comment line which causes page advance before the line is printed.

Compiler Directing Statement

A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

Complex Condition

A condition in which one or more logical operators act upon one or more conditions.

Computer-Name

A system-name that identifies the computer upon which the program is to be compiled or run.

Condition

A status of a program at execution time for which a truth value can be determined. Where the term "condition" (condition-1, condition-2, ...) appears in these language specifications in or in reference to "condition" (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-Name

A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of a task switch or a user switch.

Condition-Name Condition

Causes a conditional variable to be tested to see whether its value matches any of the values belonging to a condition-name.

Conditional Expression

A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement.

Conditional Statement

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

Conditional Variable

A data item with one or more values to which a condition-name is assigned.

Connective

A reserved word that is used to:

1. Associate a data-name, paragraph-name, condition-name, or text-name with its qualifier.
2. Link two or more operands written in a series.
3. Form conditions (logical connectives). (See Logical Operator.)

Contiguous Items

Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

Control Break

A change in the value of a data item that is referenced in the CONTROL clause.

More generally, a change in the value of a data item that is used to control the hierarchical structure of a report.

Control Break Level

The relative position within a control hierarchy at which the most major control break occurred.

Control Data Item

A data item, a change in whose contents may produce a control break.

Control Data-Name

A data-name that appears in a CONTROL clause and refers to a control data item.

Control Footing

A report group that is presented at the end of the control group of which it is a member.

Control Group

A contiguous set of data assigned to a control data item within the control hierarchy.

For a given control data item, the control group consists of the entire sequence of control headings, control footings and their associated report groups.

Control Heading

A report group that is presented at the beginning of the control group of which it is a member.

Control Hierarchy

A designated sequence of report subdivisions defined by the positional order of FINAL and the data-names within a CONTROL clause.

Conversion

The implicit transformation of numerical values from one format to another, or of index values into table occurrence numbers and vice versa.

a) In the case of index values (binary numbers) and table occurrence numbers, transformation occurs according to the formula:
index value = (occurrence number - 1) * length of table item
Hence conversion depends on the table used.
b) In case USAGEs vary from one numeric data item to another.

Counter

A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Currency Symbol

The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign ($).

Current Record

The record which is available in the record area associated with the file.

Current Record Pointer

A conceptual entity that is used in the selection of the next record.

Data Clause

A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Data Description Entry

An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Item

A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL program.

Data-Name

A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats, "data-name" represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

Debugging Line

A debugging line is any line with "D" in its indicator area (column 7 on the COBOL coding sheet).

Declaratives

A set of one or more sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative-Sentence

A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

Delimiter

A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Direct Indexing

With direct indexing, the index used is in the form of a direct subscript. (See Direct Subscripting.)

Direct Organization

Direct organization entails the use of relative track addressing. When this form of addressing is used, the sequence of the logical records in the file is determined by the actual key, which is made available by the user.

Direct Subscripting

With direct subscripting, the subscript is indicated either as an integral literal or as a data-name described as a numerical elementary item with no character positions to the right of the assumed decimal point.

Division

A set of zero, one or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program: Identification, Environment, Data and Procedure.

Division Header

A combination of words followed by a period and a space that indicates that beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING data-name-1 [data-name-2] ...].
```

Dynamic Access

The method of switching between sequential and random access. This method can be specified for relative or indexed files only.

Editing Character

A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	Space
0	Zero
+	Plus
-	Minus
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
/	Stroke (virgule, slash)

Elementary Item

A data item with no further logical subdivisions.

Entry

Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

Entry-Name

The name by which an entry point in a program called in the COBOL source program language is known to the operating system. This name is defined by means of an ENTER statement, and also occurs in a CALL statement within the calling program.

Execution Time

(See Object Time)

Extend Mode

The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Extended Access

A method of switching to and from sequential and random access. This access method may only be specified for indexed files.

Figurative Constant

A compiler generated value referenced through the use of certain reserved words, or a user-defined constant that may be referenced by user-assigned names.

File

A collection of records.

File Clause

A clause that appears as part of any of the following Data Division entries:

File description (FD)
Sort-merge file description (SD)
Report description (RD)

File Description Entry

An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name

A user-defined word that names a file described in a file description entry or a sort-merge file description entry within the File Section of the Data Division.

File Organization

The permanent logical file structure established at the time that a file is created.

Format

A specific arrangement of character strings and separators within a statement or clause.

Group Item

A named contiguous set of elementary or group items.

High Order End

The leftmost character of a string of characters.

I-O Mode

The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

I-O Status

A value moved to a two-character data item to inform the COBOL program of the status of an input/output operation. This value is only moved when the FILE STATUS clause has been specified in the FILE-CONTROL paragraph.

Identifier

The name of a data item consisting of the data-name and, if required, a syntactically correct combination of qualifiers, subscripts and indices.

Imperative Statement

A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Implementor-Name

A name taken from the following list:

CONSOLE 1)	Job-Variable-Name
TERMINAL 1)	TSW-0 to TSW-31
SYSIN, SYSIPT or SYSRDR 1)	USW-0 to USW-31
SYSOUT 1) or SYSLST 1) or SYSLST01-SYSLST99	COMPILER-INFO
SYSPUNCH, SYSPCH or SYSOPT 1)	CPU-TIME
CSP	PROCESS-INFO
C01 to C08; C10, C11	TERMINAL-INFO
Literal	

1) = "reserved words" within the Environment Division.

Index

A computer storage area or register, the contents of which represent the identification of a particular element in a table.

Index Data Item

A data item in which the value associated with an index-name can be stored.

Index-Name

A user-defined word that names an index associated with a specific table.

Indexed Data-Name

An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed File

A file with indexed organization.

Indexed Organization

The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Indicator Area

Column 7 on the COBOL coding sheet.

Input File

A file that is opened in the input mode.

Input Mode

The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-Output File

A file that is opened in the I-O mode.

Input Procedure

A set of statements that is executed each time a record is released to the sort file.

Integer

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point.

Invalid Key Condition

A condition occurring at program execution time when a particular value for a key of a relative, indexed or direct file is invalid.

Key

A data item which identifies the location of one or more data items which serve to identify the ordering of data.

Key Word

A reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name

A system-name that specifies a particular programming language.

Level Indicator

Two alphabetic characters (FD, RD, SD, DB) that identify a specific type of file.

Level Number

A one or two digit number which, in the range 1 through 49, indicates the position of a data item in the hierarchical structure of a logical record or which, in the case of level-numbers 66, 77, and 88, identifies special properties of a data description entry.

Library-Name

A user-defined word that identifies a source program library, which may contain more than one COBOL text with various names.

Library Text

A sequence of character-strings and/or separators in a COBOL library.

Line

(See Report Line)

Line Number

An integer that denotes the vertical position of a report line on a page.

Literal

A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator

One of the reserved words AND, OR, or NOT. In the formation of a compound condition, AND and OR can be used as logical connectives. NOT can be used for logical negation.

Logical Record

A data item at the highest level in the hierarchy (level-number 01) which does not occur in any other record.

Low Order End

The rightmost character of a string of characters.

Mass Storage

A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage File

A collection of records that is assigned to a mass storage medium.

Merge File

A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

Mnemonic-Name

A user-defined word that is associated in the SPECIAL-NAMES paragraph of the Environment Division with a specified implementor-name.

Native Character Set

The EBCDIC character set.

Native Collating Sequence

The collating sequence defined in the EBCDIC character set.

Negated Combined Condition

The "NOT" logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition

The "NOT" logical operator immediately followed by a simple condition.

Next Executable Sentence

The next sentence to which control will be transferred after execution of the current statement is complete.

Next Executable Statement

The next statement to which control will be transferred after execution of the current statement is complete.

Next Record

The record which logically follows the current record of a file.

Noncontiguous Items

Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.

Nonnumeric Item

A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal

A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set except the quotation mark. To represent quotation marks within a nonnumeric literal, the ' (apostrophe) must be used.

Numeric Character

A character that belongs to the following set of digits:
0,1,2,3,4,5,6,7,8,9.

Numeric Item

A data item whose value is represented by the digits "0" through "9".

Its sign, if required, must be represented by a permitted form of "+" or "-".

Numeric Literal

A literal composed of one or more numeric characters that also may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

Object Program

A set of executable machine language instructions representing the result of the operation of a COBOL compiler on a source program together with a linkage run.

Object Time

The time at which an object program is executed.

Open Mode

The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file.

In particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

Operand

Whereas the general definition or operand is "that component which is operated upon", for the purposes of this publication, any lowercase word (or words) that appears in a statement, paragraph, clause or entry format may be considered to be an operand.

Operational Sign

An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional Word

A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File

A file that is opened in either the output mode or extend mode.

Output Mode

The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.

Output Procedure

A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merged order.

Page

A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

Page Body

That part of the logical page in which lines can be written and/or spaced.

Page Footing

A report group that is presented at the end of a report page and is output prior to page advance whenever this is caused by a page advance condition.

Page Heading

A report group that is presented at the beginning of a report page and is output immediately after page advance whenever this is caused by a page advance condition.

Paragraph

In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header

A reserved word placed above the paragraphs in the Identification and Environment Divisions for identification purposes.
The permissible paragraph headers are:

In the Identification Division:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
REMARKS.
```

In the Environment Division:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name

A user defined word that identifies a paragraph in the Procedure Division. Paragraph-names must begin in column 8.

Phrase

A phrase is a ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record

(See Block)

Printable Group

A report group that contains at least one print line.

Printable Item

A data item, the extent and contents of which are specified by an elementary report entry. This elementary report entry contains a COLUMN NUMBER clause, a PICTURE clause, and a SOURCE, SUM or VALUE clause.

Procedure

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-Name

A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified), or a section-name.

Program Identification Area

Columns 73 through 80 on the COBOL coding form.

Program-Name

A user-defined word that identifies a COBOL source program.

Punctuation Character

A character that belongs to the following set:

Character	Meaning
,	Comma
;	Semicolon
.	Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
␣	Space
=	Equal sign

Qualified Data-Name

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier

1. A data-name which is used in a reference together with another data name at a lower level in the same hierarchy.

2. A section-name which is used in a reference together with a paragraph-name specified in that section.

3. A library-name which is used in a reference together with a text-name associated with that library.

Random Access

An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

Record

(See Logical Record)

Record Area

A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description Entry

The total set of data description entries associated with a particular record.

Record Indicator

The record indicator is the second portion of an actual key in a directly organized file. It provides the record with a symbolic name.

Record Key

A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record Name

A user-defined word that names a record described in a record description entry in the Data Division.

Reference Format

Standard method for describing a statement format in a COBOL source program.

Relation

(See Relational Operator)

Relation Character

A character that belongs to the following set:

Character	Meaning
>	Greater than
<	Less than
=	Equal to

Relation Condition

A condition which can yield a truth value. A relation condition causes two operands to be compared. Either of these operands may be an identifier, a literal or an arithmetic expression.

Relational Operator

A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meaning are:

Character	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

Relative File

A file with relative organization.

Relative Indexing

With relative indexing, the index takes the form of a relative subscript. (See Relative Subscripting)

Relative Key

A key whose contents identify a logical record in a relative file.

Relative Organization

The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Relative Subscripting

With relative subscripting, the name of the table element is followed by a subscript in the form

```
    (data-name + integer)
or (data-name - integer).
```

Report Clause

A clause, in the Report Section of the Data Division, that appears in a report description entry or a report group description entry.

Report Description Entry

An entry in the Report Section of the Data Division that is composed of the level indicator RD, followed by a report name, followed by a set of report clauses as required.

Report File

An output file whose file description entry contains a report clause. The contents of a report file consist of records that are written under control of the Report Writer Control System.

Report Footing

A report group that is presented only at the end of a report.

Report Group

In the Report Section of the Data Division, an 01-level entry and its subordinate entries.

Report Group Description Entry

An entry in the Report Section of the Data Division that is composed of the level-number 01, the optional data-name, a TYPE clause, and an optional set of report clauses.

Report Heading

A report group that is presented only at the beginning of a report.

Report Line

A division of a page representing one row of horizontal character positions.

Report-Name

A user-defined word that names a report described in a report description entry within the Report Section of the Data Division.

Report Writer Logical Record

A record that consists of the Report Writer print line and associated control information necessary for its selection and vertical positioning.

Reserved Word

A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Routine-Name

A user-defined word that identifies a procedure written in a language other than COBOL.

Run Unit

A set of one or more object programs which function, at object time, as a unit to provide problem solutions.

Section

A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header

A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data and Procedure Division. In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

In the Data Division:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
COMMUNICATION SECTION.
REPORT SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by as segment-number (optional), followed by a period and a space.

Section-Name

A user-defined word which names a section in the Procedure Division.

Segment-Number

A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters "0", "1", ..., "9". A segment-number may be expressed either as a one or two digit number.

Sentence

A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator

A punctuation character used to delimit character-strings.

Sequence Number Area

Columns 1 through 6 on the COBOL coding form.

Sequential Access

An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor sequence determined by the order of records in the file.

Sequential File

A file with sequential organization.

Sequential Organization

A permanent logical file structure in which the records are arranged and read in the same order in which they were created.

Sign Condition

A condition which establishes whether the algebraic value of a data item or an arithmetic expression is less than, greater than, or equal to zero.

Simple Condition

Any single condition chosen from the set:

Relation condition
Class condition
Condition-name condition
Switch-status condition
Sign condition

Sort File

A collection of records to be sorted by a SORT statement.
The sort file is created and can be used by the sort function only.

Sort-Merge File Description Entry

An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

Source Program

A syntactically correct set of COBOL statements beginning at the Identification Division and ending at the end of the Procedure Division.

Special Character

A character that belongs to the following set:

Character	Meaning
+	Plus sign
-	Minus sign
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol

Special-Character Word

A reserved word which is an arithmetic operator or a relation character.

Special Registers

Compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

Statement

A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Subprogram

(See Called Program)

Subscript

An integer whose value identifies a particular element in a table or one of the data items subordinate to this element.

Subscripted Data Item

A data item in which the value associated with the data-name or condition-name can be stored.

Subscripted Data-Name

An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Sum Counter

A signed numeric data item established by a SUM clause in the Report Section of the Data Division. The sum counter is used by the Report Writer Control System in connection with summing operations.

Switch-Status Condition

A condition which indicates whether a user or task switch has been set to "on" or "off". The test is positive if the status of the switch corresponds to the setting given in the condition-name.

Symbolic Character

A user-defined word indicating a figurative constant defined by the user.

System Name

A COBOL word which is used to communicate with the operating system.

Table

A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element

A data item that belongs to the set of repeated items comprising a table.

Text Name

A user-defined word which identifies library text.

Text-Word

Any character-string or separator, except space, in a COBOL library.

Track Identifier

The first portion of an actual key for a directly organized file. It defines the track or region (relative to the start of the file) at which to begin the search for a record, or the search for free memory space for a record.

Truth Value

The representation of the result of the evaluation of a condition in terms of one of two values, "true" or "false".

Unary Operator

A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1 respectively.

Unit

A module of random access storage.

User-Defined Word

A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable

A data item whose value may be changed during execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb

A COBOL word that causes an action to be taken by the COBOL compiler or object program.

Word

A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

1.3 NOTATION

1. Definition of a format

The fixed sequence of elements within a clause or statement is referred to as a "general format". A clause or statement may be composed of various element types.

Wherever several types of sequence are followed in a clause or statement, the general format is subdivided into formats, numbered from 1 onwards. Note that the clauses must be used in the same sequence in which they are specified in the general format. In certain exceptional cases, departures from this rule are allowed. These cases, however, are identified as such.

The proper use of formats, the necessary application prerequisites and the restrictions on their use are expressed in the form of rules.

2. Elements

Clauses or statements may be constructed from the following element types:

- words in upper case
- words in upper and lower case
- level numbers
- brackets
- braces
- connectives
- special characters

3. Words

Notation	Meaning
Upper case	A word specially reserved for COBOL.
Upper case, underlined	This word must be specified by the programmer at the location given in the format. It is a COBOL keyword.
Upper case, not underlined	This word may be specified by the programmer at the location given in the format or it may be omitted. It is an optional COBOL word.
Lower case	Generic term used to represent COBOL words, literals, picture-strings, comments, or a complete syntactical unit. It must be entered by the programmer at the location given in the format. If more than one generic term of the same kind occurs in the same format, an appended number or letter is used to uniquely qualify that term for the descriptions.

An entry consisting of one or more words in upper case followed by the words "clause" or "statement" designates a clause or statement described elsewhere in this manual.

4. Level indicators and level numbers

Level indicators and level numbers which occur in the format must be supplied at the appropriate point in the COBOL source program. This manual uses the form 01, 02, ..., 09 to indicate level numbers 1, 2, ..., 9.

5. Separators

All punctuation marks, including space characters, arithmetic operators, relational operators, and all special characters, are to be used exactly as shown in the format.

Commas and semicolons are not generally shown in the formats discussed in this manual. To improve readability, they may optionally be used for delimiting a series of clauses.

Exceptions and additional rules regarding separators can be found in this chapter under the heading "Glossary".

6. Brackets []

A format specification placed in square brackets may be supplied or omitted at the option of the user. If two or more items are stacked within brackets, one or none of them may be specified.

7. Braces {}

If two or more items are stacked within braces, one of the enclosed items is required. If there is only one item, the braces perform only a combining function for a subsequent ellipsis (repetition symbol).

8. Parenthesis ()

Format items appearing within parentheses refer to table item numbers which must be specified in order to differentiate the various items in a table.

9. Ellipsis...

An ellipsis appearing in the text indicates that one or more words have been omitted without altering the meaning of the text.

An ellipsis appearing in the format indicates that the immediately preceding unit may, if desired, be repeated any number of items after it has been specified once. A repeatable unit is either a single word or a group of words combined by brackets or braces. In the latter case, the ellipsis immediately follows the closing bracket or brace; the related opening bracket or brace determines the beginning of the unit to be repeated.

10. Space -

When used in examples and tables, this character refers to a space.

11. Special characters in formats

If the characters "+", "-", ">", "<", "=" appear in a format, they must be entered whenever the format is used. This applies even if these special characters are not underlined.

12. Gray and orange shading

Gray shading indicates extensions to Standard, orange shading refers to language elements no longer supported by future COBOL standards. These orange-shaded language elements, as well as those Siemens-specific extensions dropped from the manual as "obsolete", are reported by COB1 as 0 errors in the error class "Notes".

Example 1

```
          (2)              (2) (3)          (3)
   (1)   {identifier-1 }       [identifier-2]   (4)
ADD      {           }       [             ] ...
         {literal-1 (5)}       [literal-2    ]

                      (5)
         TO identifier-m [ROUNDED] [identifier-n [ROUNDED]]...

          (6)                                (7)
         [ON SIZE ERROR  imperative statement]
```

(1)	COBOL keyword:	The indicated form is mandatory.
(2)	Brace:	One of these items must be chosen.
(3)	Bracket:	One of these items may be chosen.
(4)	Ellipsis:	The preceding item may be repeated any number of times.
(5)	Qualification:	An appended number or letter is used to avoid duplication of names.
(6)	Optional word:	The word may be omitted, or specified for the sake of clarity.
(7)	Lowercase words:	These must be entered by the programmer.

The following language elements are valid ADD statements derived from the above format; commas and semicolons are included for better readability.

```
ADD I TO J
ADD I-1, I-2, I-3 TO I-4 ROUNDED
ADD 1 TO I-1, I-2 ROUNDED, I-3
ADD I-1 TO I-2; SIZE ERROR
    GO TO ADD-ERR.
```

1.4 LANGUAGE CONCEPTS

1.4.1 COBOL Character Set

The basic linguistic unit is the character. The COBOL character set consists of 77 characters, including 26 uppercase letters, 26 lowercase letters, 10 digits, the space character, and 14 special characters.

Character	Meaning
0, 1, ..., 9	Digit
A to Z, a to z	Letters
	Space
+	Plus
-	Minus (hyphen)
*	Asterisk
/	Slash
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark 1)
(	Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than

Table 1-1 COBOL character set

1) The predefined COBOL quotation mark is the double quote ("). To allow the compilers to formally accept old COBOL programs where the quotation mark is represented by the single quote or apostrophe ('), it is necessary to use a special operand as a compile-time parameter (for details see relevant section in the "COB1 User's Guide" [7]).

When nonnumeric literals, comments or comment lines are used, the character set is extended to comprise the whole set of characters of the data processing system.

The characters which are permitted for use with each type of string and as separators (delimiters) are defined in the subsections to follow.

1.4.2 Language Structure

A character-string is a series of contiguous characters which form a COBOL word, a literal, a picture-string (in a PICTURE clause), or a comment-entry. It is delimited by separators.

Separators, COBOL words, and literals are discussed below. For the description of the picture-string, see "PICTURE Clause"; for the description of comments, see "Identification Division".

1.4.2.1 Separators

A separator consists of punctuation marks, arithmetic operators, or relational operators. These characters are listed in Table 1-2 which follows.

Character Type	Character	Meaning
Punctuation characters		Space
	,	Comma
	;	Semicolon
	.	Period
	"	Quotation mark 1)
	(	Left parenthesis
	)	Right parenthesis
Arithmetic operators	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	**	Exponentiation
Relational operators	>	Greater than
	<	Less than
	=	Equal to

Table 1-2 Punctuation marks, arithmetic operators, and relational operators

1) The prescribed representation of quotations marks in COBOL is the "double quotes" sign ("). To enable the compiler to formally accept old COBOL programs in which the quotation mark is represented by the apostrophe ('), a special operand must be used as a compile-time parameter (for details see relevant section in the "COB1 User's Guide" [7]).

Writing a correct COBOL source program requires that the following general rules for the use of punctuation marks and spaces be observed:

1. As many spaces may be used as desired, but at least one space must be used as a separator.

2. Comma, semicolon, and period, if immediately followed by a space are considered separators. They need not be preceded by a space.

   In certain formats the comma and the semicolon are indicated as separators (delimiters). When specified, they are optional and interchangeable. They may be used to achieve better readability in separating a series of clauses, a series of consecutive operands or phrases in a statement, or a series of statements.

A period must be used exactly as shown in the formats or prescribed by the rules.

3. Left and right parentheses must occur in pairs only, to delimit indices, subscripts, arithmetic expressions, or conditions. Parentheses may or may not be preceded and/or followed by spaces.

4. An opening quotation mark must be immediately preceded by a space character or a left parenthesis; a closing quotation mark must be immediately followed by one of these separators: space, comma, semicolon, period, or right parenthesis.

   Quotation marks must occur in pairs only, to delimit nonnumeric literals except where the literal is being continued.

5. According to Standard, any arithmetic or relational operator must always by preceded and followed by a blank.
   However, the compiler permits all of these operators, except the addition and subtraction operators, to be used without being enclosed in blanks. The subtraction operator (-) must always be preceded and followed by a blank. The addition operator (+) must be followed by a blank when it is preceded by an unsigned numeric literal. Both operators may be immediately preceded or followed by a parenthesis.
   A unary operator, according to the Standard, must always be preceded by a space character or a left parenthesis and, if preceding a data-name, must be followed by a space unless it precedes a numeric literal (a unary operator is a character "+" or "-" with a single operand, indicating an algebraic sign; e.g. -5).

6. The above rules are not applicable to picture-strings in PICTURE clauses, nonnumeric literals, comments, or comment lines.

1.4.2.2 Character-Strings

A character-string is a character or series of contiguous characters which forms a COBOL word, a literal, a picture-string, or a comment-entry. A character-string is delimited by separators.

1.4.2.3 COBOL Words

A word consists of 1-30 characters from the following set:

A-Z, a-z,
0-9,
- (hyphen)

A word may neither begin nor end with a hyphen, nor may it contain space characters.

Words are divided into three categories: user-defined words, system names, and reserved words.

1. User-defined words

A user-defined word is a COBOL word to be supplied by the programmer according to the format for a clause or statement. It refers to particular units of data at object time. The following subsections describe the types of user-defined words employed in COBOL programs, and state the rules for writing these names.

The 16 types of user-defined words are listed and defined in Table 1-3.

All user-defined words except segment-numbers and level-numbers must be made unique. Either there must be no other user-defined word in the source program with the same sequence of characters and punctuation marks, or the word must be qualified.

With the exception of paragraph-name, section-name, level-number, and segment-number, all user-defined words must contain at least one alphabetic character. Segment-numbers or level-numbers may be identical to other segment-numbers or level-numbers, or to paragraph-names and section-names.

Name	Meaning
Alphabet-name	An alphabetical name located in the SPECIAL-NAMES paragraph of the Environment Division and connected with with a character set and/or collating sequence.
Condition-name	The name assigned to a specific value, set of values, or range of values which an elementary data item may assume (hence, a condition of the data item). A condition-name is defined by an 88-level entry in the File, Linkage or Working-Storage Section.
Class-name	A name entered by the user in the CLASS clause of the SPECIAL-NAMES paragraph in the Environment Division to define a character set. This class-name can be referenced in the class condition.
Data-name	A name identifying a data item in the Data Division. A data-name is defined by its appearance in a data description entry.
	A special data-name is an index data-name designating an index data item. An index data item is a data item whose description contains the USAGE IS INDEX clause.
File-name	A name assigned to a set of input data or output data. A file-name is defined by its appearance in the SELECT clause of the FILE CONTROL paragraph and its use as the name of an FD entry.
	A special file-name is a sort-file-name that names a sort-file. A sort-file-name is defined by it's appearance in the SELECT clause of the FILE CONTROL paragraph and its use to name an SD entry in the File Section.
Index-name	A name of an index for a particular table. An index-name is declared by its occurrence in the INDEXED BY phrase of the OCCURS clause.
Level-number	A level-number indicates the position of a data item in the hierarchical structure of a record or indicates special properties of a data description entry. Level-numbers are defined by their appearance in a data description entry.
Library-name	A name of an entry in the COBOL source program library. The library may contain more than one text with various names.
Mnemonic-name	A fixed name, provided the programmer associated it with a particular implementor-name in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.
Paragraph-name	A paragraph-name is used to name a paragraph in the Procedure Division. Paragraph-names are written starting at Area A.

Name	Meaning
Program-name	The name used to identify the program. The program-name is defined by its used in the PROGRAM-ID paragraph of the Identification Division. It may also appear in a CALL Statement of a corresponding calling program.
Record name	The name of a data record. A data record is declared by a 01-level entry in the FILE SECTION, LINKAGE SECTION, WORKING-STORAGE SECTION or SUB-SCHEMA SECTION.
Report-name	The name of a report. A report-name is defined by its occurrence in the REPORT clause of an FD entry; it is used to name an RD entry in the Report Section.
Section-name	A section-name is used to name a section in the Procedure Division. A section-name is written starting at Area A and is followed by the word SECTION.
Segment-number	A number to classify sections in the Procedure Division for purposes of segmentation. It is defined by its use in a section header.
Symbolic-character	A name for a figurative constant defined by the user in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph.
Text-name	Name of an entry in the COBOL source program library. The entry is copied from the library by the COPY statement.

Table 1-3 COBOL user-defined words

2. System-names

A system-name is a COBOL word which is used as an interface with the operating system environment. System-names are defined by the implementor and may vary from compiler to compiler. From the programmer's point of view, the system-names of a specific compiler are treated as reserved words.

The system names in the compiler include:

Computer-name in the SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs.

Implementor-name in the SPECIAL-NAMES paragraph.

Implementor-name in the ASSIGN clause.

For further details consult the pertinent descriptions.

3. Reserved words

COBOL includes a fixed number of reserved words, the COBOL words, (see Table 1-7, "Reserved COBOL Words").

A reserved word serves a specific purpose and must be used only in the context specified in the formats; it may not occur in the source program as a user-defined word or system-name.

A complete list of reserved words is supplied in Table 1-7. All reserved words marked with * in Table 1-7 are treated as reserved words only if DML (Data Manipulation Language) statements are being used for compilation; otherwise they may be employed as user-defined words. Compilation with DML statements occurs when

```
SUB-SCHEMA SECTION
```

is specified (see the "UDS Reference Manual" [6]).

There are six types of reserved words:

a) Keywords
b) Optional words
c) Connectives
d) Special register
e) Figurative constants
f) Special-character words

a) Keywords

A keyword is a reserved word which must be present if the format in which the word is used occurs in the source program. Within each format such words appear in upper case and are underscored. Keywords are only allowed in the formats mentioned.

There are three types of keywords:

1. Verbs such as ADD, READ, and CALL.
2. Required words which are encountered in statement and entry formats.
3. Words which have a specific functional significance, such as NEGATIVE, SECTION, etc.

Some keywords may be abbreviated (e.g. PIC for PICTURE).

b) Optional words

Within each format, words written in upper case which are not underscored are called "optional words". These words may be used at the option of the user. The presence or omission of an optional word in a program has no effect on the meaning of the COBOL program. However, an optional word may not be misspelled or replaced with another word.

c) Connectives

There are three types of connectives:

Category	Usage	Connective
Qualifier connective	To associate a data-name or paragraph-name with its qualifier.	OF IN
Series connective	To link two or more consecutive operands.	,(comma)
Boolean operand	To create compound conditions.	AND OR AND NOT OR NOT

Table 1-4 Types of COBOL connectives

d) Special registers

Special registers are data items in which information produced with the use of certain COBOL features is stored. The attributes of these registers are predefined, and each register has a fixed name. Thus, the programmer does not have to define these registers in the Data Division. The ten special registers are listed in Table 1-5.

Register name	Description	Use
TALLY	5-digit unsigned data item with COMPUTATIONAL phrase (see "USAGE Clause").	TALLY may be used wherever a data item with an integral value can occur. If, e.g., the current value of TALLY is 3, the following statements are equivalent: ADD 3 TO ALPHA. ADD TALLY TO ALPHA. The main use of TALLY is to include information created by the EXAMINE statement (see "EXAMINE Statement").
LINE-COUNTER PAGE-COUNTER PRINT-SWITCH CBL-CTR	Used by the Report Writer (see chapter 8, "Report Writer").	See chapter 8, "Report Writer".
LINAGE-COUNTER	A 4-byte data item containing an unsigned integer whose value is less than or equal to integer-1 or the data item referenced by data-name-1 in the LINAGE clause	A LINAGE-COUNTER register is generated by the compiler for each file whose file description entry contains a LINAGE-clause (see "LINAGE clause").
RETURN-CODE	4-digit signed data item with COMPUTATIONAL phrase and SYNCHRONIZED phrase (corresponds to PIC S9(4) COMP SYNC).	This data item exists only once for each program system. The user can use this item to exchange information between COBOL modules which were compiled separately but linked into a single object program. If the contents of the RETURN-CODE special register is not 0 when the program system terminates, a dump is created and the operating system is informed that the program terminated abnormally.
SORT-RETURN SORT-FILE-SIZE SORT-CORE-SIZE SORT-MODE-SIZE	Used by the sort section (see chapter 10, "Sort/ Merge Feature").	See chapter 10, "Sort/ Merge Feature".

Table 1-5 COBOL special registers

e) Figurative constants

The values of figurative constants are produced by the compiler and are indicated by the reserved words listed in Table 1-6. Figurative constants must not be enclosed in quotation marks. The singular and plural forms of a figurative constant are equivalent and may be used optionally.

The figurative constant [ALL] symbolic-character stands for one or more of the characters specified as the value of symbolic-character in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph (q.v.).

If a figurative constant represents a string of one or more characters, the compiler determines the length of the string according to the following rules:

1. If a figurative constant is specified in a VALUE clause or associated with another data item (e.g. moved to or compared with another data item), it is first duplicated to the right until the resultant string has at least as many character positions as the other data item.

   If this character string has more character positions than the other data item following the duplication operation, the extra positions will be truncated from the right.

   Extension or truncation of the character string of figurative constants takes place prior to and independently of any application of the JUSTIFIED clause to the other data item.

2. The character string always has a length of 1 whenever the figurative constants ZERO, SPACE, HIGH-VALUE, LOW-VALUE and QUOTE (including their plurals) are not brought into contact with another data item, particularly whenever they occur as operands in a DISPLAY, STOP, STRING or UNSTRING statement.

3. If the figurative constant ALL literal is not brought into contact with another data item, the length of the character string is equal to the length of literal.

A figurative constant can be used wherever literal occurs in a format, except for the following cases:

1. If literal is restricted to numeric literals, the only figurative constant allowed is ZERO (ZEROS, ZEROES).

2. The figurative constant ALL literal cannot be brought into contact with numeric or numeric-edited data items.

3. Apart from its use in the figurative constant ALL literal, the word ALL has no function; it serves only to enhance readability.

If the figurative constants HIGH-VALUE[S] or LOW-VALUE[S] are used in a source program (except for the ALPHABET clause), the character currently associated with this constant is dependent on the collating sequence defined for the program and belonging to the character set (see "OBJECT-COMPUTER paragraph" and "SPECIAL-NAMES paragraph").

Each reserved word used to assign a value to a figurative constant constitutes a character string of its own; if the word ALL is used it constitutes two character strings.

If alphabet-name-2 is specified in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph or in the CODE-SET clause of a data

description entry (see "CODE-SET clause"), the character code type is defined by the ALPHABET clause.

If the IN phrase is omitted, symbolic-character-1 stands for the character whose position within the collating sequence of the hardware-specific character set is indicated by integer-1.

If the IN phrase is used, integer-1 refers to the character set named by alphabet-name-2.

The internal representation of symbolic-character-1 is identical to that of the corresponding character in the hardware-specific character set.

Table 1-6 lists the figurative constants and indicates the values they represent.

Figurative constant	Corresponding value	Examples 1)
[ALL]ZERO or [ALL]ZEROS or [ALL]ZEROES	One or more occurrences of the character 0 (X'F0') or binary zero (X'00'), depending on the description of the data item.	Statement: MOVE ZEROS TO ALPHA. Contents of ALPHA: If ALPHA is a binary item: X'00000000' If ALPHA is an external decimal item: X'F0F0F0F0'(= C'0000') If ALPHA is an internal decimal item: X'0000000F'.
[ALL] SPACE or [ALL] SPACES	One or more occurrences of the character space (X'40').	Statement: MOVE SPACE TO ALPHA. Contents of ALPHA: X'40404040'(= C'␣␣␣␣')
[ALL] HIGH-VALUE or [ALL] HIGH-VALUES	With COLLATING SEQUENCE unspecified: One or more occurrences of the character that has the highest value in the EBCDIC collating sequence (X'FF').	Statement: MOVE HIGH-VALUE TO ALPHA. Contents of ALPHA: X'FFFFFFFF'(=C'◇◇◇◇')
	With COLLATING SEQUENCE specified: The character with the highest position in the program collating sequence.	Entry in SPECIAL-NAMES paragraph: ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 194. The highest position belongs to the character at the 194th position of the EBCDIC character set, i.e. the character A. A is assigned to HIGH-VALUE.
[ALL] LOW-VALUE or [ALL] LOW-VALUES	With COLLATING SEQUENCE unspecified: One or more occurrences of the character that has the lowest value in the EBCDIC collating sequence (X'00').	Statement: MOVE LOW-VALUES TO ALPHA. Contents of ALPHA: X'00000000'

Figurative constant	Corresponding value	Examples 1)
[ALL] LOW-VALUE or [ALL] LOW-VALUES (continued)	With COLLATING SEQUENCE specified: The character with the lowest position in the program collating sequence.	Entry in SPECIAL-NAMES paragraph: ALPHABET ALPHATAB IS "0" "1" "2". The lowest position belongs to the character 0. 0 is assigned to LOW-VALUE.
[ALL] QUOTE or [ALL] QUOTES	One or more occurrences of the apostrophe character (X'7F'). Note: The word QUOTE (QUOTES) cannot be used in place of an apostrophe to enclose a nonnumeric literal.	Data Division Entry: 02 BETA PIC X VALUE QUOTE. Contents of BETA: X'7F' or X'7D', depending on the current quotation mark (see COBOL character set).
ALL literal	One or more occurrences of the string of characters composing the literal. The literal must be nonnumeric.	Statement: MOVE ALL "A" TO ALPHA. Contents of ALPHA: C'AAAA' Statement: MOVE ALL "12" TO ALPHA. Contents of ALPHA: C'1212' Statement: MOVE ALL "ABC" TO ALPHA. Contents of ALPHA: C'ABCA'
[ALL] symbolic-character	One or more repetitions of the character specified as the value of symbolic-character in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph.	Description: SYMBOLIC C0 IS 193 Statement: MOVE ALL C0 TO ALPHA. Contents of ALPHA: X'C0C0C0C0'

Table 1-6 COBOL figurative constants and values

1) In these examples, assume that ALPHA, unless otherwise specified, is a 4-byte area with the data format DISPLAY.

f) Special character words

Arithmetic operators and relation characters are reserved words (see "Glossary").

Table 1-7 Reserved COBOL words 1)

<	COMP	END-DIVIDE
<=	COMP-1	END-EVALUATE
+	COMP-2	END-IF
*	COMP-3	END-MULTIPLY
**	COMP-4	END-OF-PAGE
-	COMPUTATIONAL	END-PERFORM
/	COMPUTATIONAL-1	END-READ
>	COMPUTATIONAL-2	END-RECEIVE
>=	COMPUTATIONAL-3	END-RETURN
=	COMPUTATIONAL-4	END-REWRITE
ACCEPT	COMPUTE	END-SEARCH
ACCESS	CONFIGURATION	END-START
ACTUAL	* CONNECT	END-STRING
ADD	CONTAINS	END-SUBTRACT
ADVANCING	CONTENT	END-UNSTRING
AFTER	CONTINUE	END-WRITE
ALL	CONTROL	ENDING
ALPHABET	CONTROLS	ENTER
ALPHABETIC	CONVERTING	ENTRY
ALPHABETIC-LOWER	COPY	ENVIRONMENT
ALPHABETIC-UPPER	CORR	EOP
ALPHANUMERIC	CORRESPONDING	EQUAL
ALPHANUMERIC-EDITED	COUNT	* ERASE
ALSO	CREATING	ERROR
ALTER	CURRENCY	ESI
ALTERNATE	* CURRENT	EVALUATE
AND	CURRENT-DATE	EVERY
ANY	CYL-OVERFLOW	EXAMINE
APPLY	CYLINDER-OFLO	EXCEPTION
ARE	DATA	* EXCLUSIVE
AREA	* DATABASE-EXCEPTION	EXHIBIT
AREAS	* DATABASE-KEY	EXIT
ASCENDING	DATE	EXTEND
ASSIGN	DATE-COMPILED	EXTENDED
AT	DATE-WRITTEN	EXTENDED-SEARCH
AUTHOR	DAY	EXTERNAL
BEFORE	DAY-OF-WEEK	FALSE
BEGINNING	* DB	FD
BINARY	DE	* FETCH
BLANK	DEBUGGING	FILE
BLOCK	DECIMAL-POINT	FILE-CONTROL
BLOCK-DENSITY	DECLARATIVES	FILE-LIMIT
BOTTOM	DELETE	FILE-LIMITS
BY	DELIMITED	FILLER
CALL	DELIMITER	FINAL
CANCEL	DEPENDING	* FIND
CBL-CTR	DESCENDING	* FINISH
CD	DESTINATION	FIRST
CF	DETAIL	FOOTING
CH	DIRECT	FOR
CHANGED	DISABLE	FORM-OVERFLOW
CHARACTER	DISC	* FREE
CHARACTERS	* DISCONNECT	FROM
CHECKING	DISPLAY	GENERATE
CHECKPOINT	DIVIDE	* GET
CLASS	DIVISION	GIVING
CLOCK-UNITS	DOWN	GLOBAL
CLOSE	* DUPLICATE	GO
COBOL	DUPLICATES	GREATER
CODE	DYNAMIC	GROUP
CODE-SET	EGI	HEADING
COLLATING	ELSE	HIGH-VALUE
COLUMN	EMI	HIGH-VALUES
COMMA	* EMPTY	I-O
COMMIT	ENABLE	I-O-CONTROL
COMMON	END	ID
COMMUNICATION	END-ADD	IDENTIFICATION
CONSOLE	END-CALL	IF
CSP	END-COMPUTE	IN
C01 ... C12	END-DELETE	* INCLUDING

1) The meaning of the character * is explained under "Reserved words" and at the end of this table.

Table 1-7 Reserved COBOL words (continued)

INDEX	OR	RESULT
INDEXED	ORDER	* RETAINING
INDICATE	ORGANIZATION	* RETRIEVAL
INITIAL	OTHER	RETURN
INITIALIZE	OTHERWISE	RETURN-CODE
INITIATE	OUTPUT	REVERSED
INPUT	OVERFLOW	REWIND
INPUT-OUTPUT	* OWNER	REWRITE
INSPECT	PACKED-DECIMAL	RF
INSTALLATION	PADDING	RH
INTO	PAGE	RIGHT
INVALID	PAGE-COUNTER	ROLLBACK
IS	PERCENT	ROUNDED
JUST	PERFORM	RUN
JUSTIFIED	* PERMANENT	SAME
* KEEP	PF	SD
KEY	PH	SEARCH
LABEL	PIC	SECTION
LAST	PICTURE	SECURITY
LEADING	PLUS	SEGMENT
LEFT	POINTER	SEGMENT-LIMIT
LENGTH	POSITION	SELECT
LESS	POSITIONING	* SELECTIVE
LIMIT	POSITIVE	SEND
* LIMITED	PRINT-SWITCH	SENTENCE
LIMITS	PRINTING	SEPARATE
LINAGE	* PRIOR	SEQUENCE
LINAGE-COUNTER	PROCEDURE	SEQUENTIAL
LINE	PROCEDURES	SET
LINE-COUNTER	PROCEED	* SET-SELECTION
LINES	PROCESSING	* SETS
LINKAGE	PROGRAM	SIGN
LOCK	PROGRAM-ID	SIZE
LOG	* PROTECTED	SORT
LOW-VALUE	PURGE	SORT-CORE-SIZE
LOW-VALUES	QUEUE	SORT-FILE-SIZE
* MASK	QUOTE	SORT-MERGE
* MEMBER	QUOTES	SORT-MODE-SIZE
* MEMBERS	RANDOM	SORT-RETURN
* MEMBERSHIP	RD	SORT-TAPE
MEMORY	READ	SORT-TAPES
MERGE	* READY	SOURCE
MESSAGE	* REALM	SOURCE-COMPUTER
MODE	* REALM-NAME	SPACE
* MODIFY	RECEIVE	SPACES
MODULES	RECORD	SPECIAL-NAMES
MORE-LABELS	RECORD-OVERFLOW	STANDARD
NAMED	REEL	STATUS
NATIVE	REFERENCE	STOP
NEGATIVE	REFERENCES	* STORE
NEXT	RELATIVE	STRING
NO	RELEASE	SUB-QUEUE-1
NOMINAL	REMAINDER	SUB-QUEUE-2
NOT	REMARKS	SUB-QUEUE-3
NUMBER	REMOVAL	* SUB-SCHEMA
NUMERIC	RENAMES	SUBTRACT
NUMERIC-EDITED	REORG-CRITERIA	SUM
OBJECT-COMPUTER	REPEATED	SUPPRESS
* OCCURRENCE	REPLACE	SYMBOLIC
OCCURS	REPLACING	SYNC
OF	REPORT	SYNCHRONISED
OFF	REPORTING	SYSLST
OMITTED	REPORTS	SYSLST01 ... SYSLST99
ON	RERUN	SYSOPT
* ONLY	RESERVE	SYSPCH
OPEN	RESET	SYSPUNCH
OPTIONAL	RESTRICTED	SYSRDR

1) The meaning of the character * is explained under "Reserved words" and at the end of this table.

Table 1-7 Reserved COBOL words (continued)

* SYSTEM	TOP	* USAGE-MODE
TABLE	TRACE	USE
TALLY	TRACK-AREA	USING
TALLYING	TRACKS	USW-0 ... USW-31
TAPE	TRAILING	VALUE
TAPES	TRANSFORM	VALUES
* TENANT	TRUE	VARYING
TERMINAL	TRY	* VIA
TERMINATE	TSW-0 ... TSW-31	WHEN
TEST	TYPE	WITH
TEXT	UNIT	* WITHIN
THAN	UNITS	WORDS
THEN	UNSTRING	WORKING-STORAGE
THROUGH	UNTIL	WRITE
THRU	UP	WRITE-ONLY
TIME	* UPDATE	WRITE-VERIFY
TIMES	UPON	ZERO
TO	UPSI-0 ... UPSI-7	ZEROES
TODAYS-DATE	USAGE	ZEROS

* Words marked with an asterisk are treated as reserved words only if compilation occurs with DML statements (DML = Data Manipulation Language). Otherwise they can be used as user-defined words. Compilation proceeds with DML statements whenever SUB-SCHEMA SECTION is specified

4. Literals

A literal is a character-string whose value is determined by the very characters which form the literal; alternatively, the string may represent a reserved word which corresponds to a figurative constant. Literals are either numeric or nonnumeric.

a) Nonnumeric literals

A nonnumeric literal is a character-string consisting of from 1 to 180 characters, enclosed in quotes. The literal may contain any characters from the EBCDIC character set, except quotation marks. The value of the nonnumeric literal in the object program is the sequence of individual characters itself, without delimiting quotes. To represent quotation marks within a literal, the apostrophe (') or double quotes (") must be used.

Example 2

```
"CHARACTER"
"153.78"
"ADAM 'BDAM' CDAM"
```

Example 3

```
'CHARACTER'
'153.78'
'ADAM "BDAM" CDAM'
```

In other words, apostrophes are permitted within character-strings enclosed in quotation marks, and vice versa.

b) Numeric literals

There are two types of numeric literals: fixed-point literals and floating-point literals.

The value of the numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric.

- **Numeric fixed-point literals**

A fixed-point numeric literal is a string of characters chosen from the following set: the digits 0-9, the plus sign, the minus sign, and the decimal point.

Fixed-point numeric literals must be formed according to the following rules:

1. The literal may contain 1 to 18 digits.
2. The literal may contain only one sign character. If a sign is used, it must be the leftmost character of the literal. An unsigned literal is assumed to be positive.
3. The literal may contain only one decimal point. The decimal point may appear anywhere in the literal, except as the rightmost character. A decimal point designates an assumed decimal point location. (The assumed decimal point in any

numeric literal or data item is the position where the compiler and the generated object program assume the decimal point to be, though no internal memory position is reserved for a separate decimal point character.) A literal with no decimal point is an integer.

The term **integer** is used to describe a numeric literal which is unsigned and greater than zero and which has no character positions to the right of the assumed decimal point.

Example 4

(Here, the character V depicts an assumed decimal point.)

Literal	Location of assumed point	Internal sign	No. of digit positions assigned
+123	123 V	+	3
3.765	3V765	+	4
-45.7	45V7	-	3

- **Numeric floating-point literals**

A numeric floating-point literal is a data item whose potential value range is too large to be represented with a fixed point.

A numeric floating-point literal must have the following format:

mantissa exponent

The mantissa consists of an optional sign followed by 1 to 18 digits with a decimal point. The decimal point may be specified anywhere in the mantissa.

The exponent consists of the symbol E, followed by an optional sign and then by one or more digits (the exponent 0 can be written as 0 or 00).

The literal must not contain blanks. The exponent must be specified immediately to the right of the mantissa.

The sign is the only optional character in the format. An unsigned mantissa or an unsigned exponent is interpreted as positive.

The value of the literal is the product of the mantissa and the power of 10 given by the exponent.

The absolute value of a number represented by a floating-point literal must not exceed $7.2*10^{75}$

Example 1-5

$+1.5E-2=1.5*10^{-2}$

The absolute value of a number represented by a floating-point literal must not exceed $7.2*10^{75}$

Figurative constants and their values are summarized in Table 1-6.

5. Picture-strings

A picture-string consists of various combinations of characters from the COBOL character set, which are used as symbols (see PICTURE clause).

Any punctuation character within a picture-string is not interpreted as a punctuation character but as a picture-string symbol.

6. Comment-entry

A comment-entry is an entry consisting of characters from the character set of the data processing system in any combination.

1.4.2.4 Concept of Computer-Independent Data Description

The data should be described as computer-independent as possible. To do this, the characteristics and attributes of the data are not described in a computer-specific format but rather in a standard data format. This standard data format is derived from general data processing applications and uses, regardless of how the DP system represents numbers internally, the decimal system to represent numbers and all remaining characters of COBOL character set to specify nonnumeric data items.

1. Concept of logical record and file

It is fundamental for the definition of a file that a distinction be made between the physical aspects of the file and the conceptual characteristics of the data contained in the file.

a) Physical aspects of a file

The physical aspects of a file describe the data as it appears on the input or output medium. They include such information as:

- The grouping of logical records, taking into account the physical boundaries of the storage medium.
- The manner in which a file may be identified.

b) Characteristics of a file

The characteristics of a file comprise the explicit definitions of each logical extent within the file itself. The input-output statements in a COBOL program refer to logical records.

It is extremely important to distinguish between a physical record and a logical record.

- A **physical data record** (or **block**) is a unit of information whose size and recording mode provide for optimum data storage on an input or output medium for a particular computer installation. The size of a physical data record is machine-oriented and permits no direct comparison with the size of the logical file information.

- A **logical record** (or simply **record**) is a group of associated data which can be uniquely designated and treated as an entity, and can be read from or stored in a file. A block may contain different kinds of records.

The term "record" is not restricted to data stored on an external data medium, but can be applied to the definition of working storage for data created internally during program execution.

In this manual, "data record" always refers to logical records.

2. Level concept

The level concept permits the structuring of a logical data record. Data processed by a COBOL program can be described as elementary items, group items, data records and files (for file description see chapters 3 to 6).

a) Elementary items

An elementary item is the smallest unit of data bearing a name, i.e. it is not divisible into further elementary items. An elementary item is described with a PICTURE clause (except in the case of description with COMP-1, COMP-2).

The length of an elementary item may not exceed 65,535 bytes.

b) Group items

Several elementary items combined form a group item. Thus, a number of elementary items may be addressed simultaneously under the name of the group item. Each group consists of an elementary item or a series of elementary items. Groups, in turn, may be combined to form two or more group items. Consequently, an elementary item may belong to more than one group item (see Fig. 1-1, Fig. 1-2). The name of a group item may not be described with the PICTURE clause.

c) Data records

A data record is a data item which is not subordinate to another data item. It consists of a group item and/or several elementary items, or it is itself an elementary item. The description of a data record must start in Area A.

d) Level numbers

Data are divided into various levels. These levels are indicated by means of level numbers. The numbers 01 to 49 are allowed as level numbers. In addition, there are special level numbers: 66, 77, and 88. In a source program, every level number must be given a separate entry.

Since a data record represents the largest organizational unit, level numbers for data records start at 01. Hierarchically subordinate items are assigned numerically higher level numbers (from 02 to 49). The level number of a subordinate data item must be greater than that of a higher-ranking data item by one or more units. Once an elementary item has been described, only those level numbers which have already appeared in the data record description entry are permitted.

Example 1-6

right:

```
01  DATA RECORD.
    05  GROUP-ITEM-1.
        10  ELEMENTARY-ITEM-11 ...
        10  ELEMENTARY-ITEM-12 ...
        10  ELEMENTARY-ITEM-13 ...
    05  GROUP-ITEM-2.
        10  ELEMENTARY-ITEM-21 ...
        10  ELEMENTARY-ITEM-22 ...
```

wrong:

```
01  DATA RECORD.
    05  GROUP-ITEM-1.
        10  ELEMENTARY-ITEM-11 ...
        10  ELEMENTARY-ITEM-12 ...
        10  ELEMENTARY-ITEM-13 ...
    03  GROUP-ITEM-2.
        10  ELEMENTARY-ITEM-21 ...
        10  ELEMENTARY-ITEM-22 ...
```

There are three types of data for which no level concept exists. These are assigned the level numbers 66, 77, and 88:

- Level number 66 is given to the names of data items described with the RENAMES clause (see "RENAMES Clause").
- Level number 77 is given to structure-independent data items of the WORKING-STORAGE SECTION or LINKAGE SECTION (see chapter 2, "77-Level Description Entry").
- Level number 88 is given to the explanation of condition-names (see "VALUE Clause").

Level numbers 01 and 77 must be located in Area A of the source program; all other level numbers may begin in Area A or Area B.

For further rules see chapter 2, "Level Numbers".

3. Data classes

The five categories of data items (see "PICTURE Clause") are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric items, the classes and categories are synonymous. The alphanumeric class includes the categories "alphanumeric edited", "numeric edited" and "alphanumeric" (without editing).

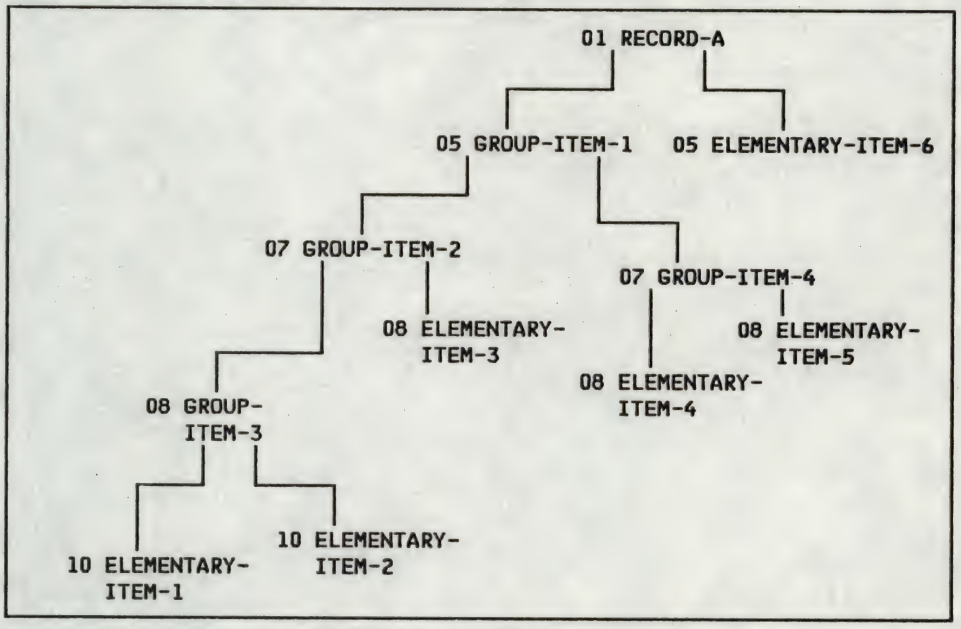


Fig. 1-1 Relationship between group items and elementary items in a data record

```
01 RECORD-A.
   05 GROUP-ITEM-1.
      07 GROUP-ITEM-2.
         08 GROUP-ITEM-3.
            10 ELEMENTARY-ITEM-1...
            10 ELEMENTARY-ITEM-2...
         08 ELEMENTARY-ITEM-3...
      07 GROUP-ITEM-4.
         08 ELEMENTARY-ITEM-4...
         08 ELEMENTARY-ITEM-5...
   05 ELEMENTARY-ITEM-6...
```

Fig. 1-2 Group items and elementary items in a data record

Figure 1-1 shows the structure of a sample record ; Figure 1-2 demonstrates how to use level numbers to represent this structure in the record description entry. In this example, GROUP-ITEM-3 and ELEMENTARY-ITEM-3 are part of GROUP-ITEM-2; similarly, GROUP-ITEM-2 and GROUP-ITEM-4 are part of GROUP-ITEM-1.

Each elementary item fits into one of the classes and also into one of the categories.

Group items are always classified as alphanumeric at object time, regardless of the class of the elementary items subordinate to them.

The following Table 1-8 illustrates the relationship between the classes and categories of data items.

4. Data categories

The following subsections provide a summary explanation and description of data items in the various categories, as shown above under "Data categories". The category of a data item is determined by the type of the PICTURE and USAGE clauses which are present in the description element (for further details see "PICTURE Clause" and "USAGE Clause").

Level of item	Class	Category
elementary	alphabetic	alphabetic
	numeric	numeric
	alphanumeric	numeric edited alphanumeric edited alphanumeric
group	alphanumeric	alphabetic numeric numeric edited alphabetic edited alphanumeric

Table 1-8 Classes and categories of elementary and group items

a) Alphabetic items

An alphabetic data item is a data item whose contents, if represented in standard format, can be any combination of the 26 letters of the alphabet, plus the blank. Each alphabetic character is stored in its own byte in working storage.

Its picture-string contains only the symbol A.

The data format of alphabetic items is always DISPLAY.

b) Numeric items

There are two types of numeric data items, fixed-point items and floating-point items. For the internal representation of numeric items, see "USAGE Clause", Table 2-9.

1. Fixed-point data items

A fixed-point data item is a numeric data item in which the operational decimal point is assumed to be present in every value or to be maintained at a fixed position relative to the beginning or end of the storage area reserved for the data item. The contents of a fixed-point data item must be comprised of the digits 0 through 9, provided the SIGN clause is not specified. If the SIGN clause is specified, the contents may contain a +, - or other representations of the sign in addition to the above-mentioned digits. If the picture-string contains an S for a fixed-point data item, the contents of the data item are treated as positive or negative depending on the operational sign. If the picture-string does not contain an S, the contents of the data item are treated as an absolute value.

Picture-strings for fixed-point items may contain the symbolic characters 9, P, S and V only.

COBOL recognizes three types of fixed-point numbers:

external decimal (USAGE IS DISPLAY)

binary (USAGE IS COMPUTATIONAL)

internal decimal (USAGE IS COMPUTATIONAL-3)

The differences between these three types are described under "USAGE Clause".

2. Floating-point data items

A floating-point data item is a numeric data item whose decimal point is movable, i.e. it can be located at various positions and is defined by specifying a base-10 exponent. The base thus raised to a particular power serves as the coefficient of a fixed-point number (mantissa), thereby representing the floating-point number.

Floating-point data items are only used for data whose potential value range is too large for fixed-point representation.

There are two kinds of floating-point data items: external floating-point data items and internal floating-point data items.

- **External floating-point data items**

  The picture-string for an external floating-point data item can contain the characters 9, . (decimal point), V, E, + and -. Each character, except V, occupies one byte of memory, and is contained in each printout (see "PICTURE Clause" and "USAGE Clause" for further details).

  The data format of an external floating-point data item is always DISPLAY.

- **Internal floating-point data items**

  There are two kinds of internal floating-point data items:

  Short precision (USAGE IS COMPUTATIONAL-1) with a length of 4 bytes and

  double precision (USAGE IS COMPUTATIONAL-2) with a length of 8 bytes.

  Both encompass the same value range. A short-precision data item permits precision of 7 decimal digits. A double-precision data-item allows precision of 16 decimal digits.

  A PICTURE clause is prohibited for internal floating-point data items; the length of this kind of data item is determined by its USAGE clause.

c) Alphanumeric items

An alphanumeric item is one whose contents, when represented in standard data format, are any characters from the EBCDIC set.

Its picture-string is restricted to combinations of the symbols A, X, and 9. The item is treated as if its picture-string contained all X's.

A picture-string which contains all A's or all 9's does not define an alphanumeric item.

The data format of an alphanumeric item is always DISPLAY.

d) Numeric edited items

A numeric edited item describes the editing of a numeric value. When a numeric edited item is a receiving field for a MOVE, the data coming into the field is edited according to the picture-string specified.

The picture-string of a numeric edited item is restricted to certain combinations of the symbols B, / (slash), P, V, Z, 0 (zero), 9, , (comma), . (decimal point), *, +, -, CR, DB and $ (currency sign). The allowable combinations are determined from editing rules and the order or precedence of symbols (see the "PICTURE Clause" for a discussion of these topics). The maximum number of digits that may be represented in a picture-string for a numeric edited item is 18.

Data is stored one character per byte. The contents of a character position that represents a digit must be one of the numerals 0 through 9.

The data format of a numeric edited data item is always DISPLAY.

e) Alphanumeric edited items

An alphanumeric edited item describes the editing of an alphanumeric value. When an alphanumeric edited item is a receiving field for a MOVE statement, the data coming into the field is edited according to the specified picture of the field.

The picture-string of an alphanumeric edited item is restricted to certain combinations of the following characters: A, / (slash), X, 9, 0 (zero), and B (see "PICTURE Clause" for further details).

The contents of an alphanumeric edited item, when represented in standard data format, are allowable characters chosen from the EBCDIC character set.

The data format of an alphanumeric edited item is always DISPLAY.

5. Algebraic signs

There are two categories of algebraic signs: operational signs, which are associated with signed numeric items and signed numeric literals to specify their algebraic properties, and editing signs, which occur e.g. in edited reports in order to indicate the sign of a data item.

Editing signs are inserted in a data item by means of the sign control character of the relevant picture-string (see "PICTURE Clause").

6. Alignment

The alignment of data within elementary data items depends on the category of the receiving field. The alignment within group items is the same as for alphanumeric receiving fields.

a) Numeric data items

If the receiving field is described as a numeric item, the data is aligned on the decimal point and is moved to the receiving character positions with zero filling or truncation, as required at the appropriate end.
If an assumed decimal point is not supplied explicitly, the item is treated as if it had an assumed decimal point immediately following its rightmost character, and is aligned as described above.

b) Numeric edited data items

If the receiving field is a numeric edited item, the data being moved to the edited field is aligned on the decimal point, with zero filling (or without, should leading zeroes have to be replaced because of editing requirements) or truncation at the appropriate end, as required in the receiving character positions of the data item.

c) Alphanumeric, alphanumeric edited, and alphabetic data items

If the receiving field is alphanumeric (other than numeric edited), alphanumeric edited, or alphabetic, then the data being sent is moved to the receiving data position as requested, aligned on the leftmost character position in the data item, with space padding or truncation on the right.

If the JUSTIFIED clause is specified for the receiving field, refer to the changed rules in the description of "JUSTIFIED Clause".

d) Data field alignment for greater object program code efficiency

It is possible to enhance processing efficiency for particular data (in arithmetic or subscripting operations) by aligning the data on natural boundaries (halfword, fullword, doubleword).

The object program requires additional machine instructions for accessing and storing data if parts of two or more data items occur between two adjacent natural boundaries or if certain natural boundaries divide a single item.

Data items whose alignment on these natural boundaries is such that they do not require additional machine instructions, are defined as "synchronized".

Alignment may be achieved by using the SYNCHRONIZED clause or by allowing for the natural boundaries and organizing the data more effectively; in the latter case, the SYNCHRONIZED clause does not need to be supplied (see "SYNCHRONIZED Clause").

For special details on the compiler, see "Alignment by Insertion of Slack Bytes".

1.4.2.5 Implementor-Dependent Representation and Alignment of Data

1. Data format

a) Standard data format

The standard format used to store data items with USAGE DISPLAY in internal memory is as follows:

Each **character position** (as specified by the picture-string) is represented by one byte.

Each **character** is internally represented by the appropriate code from the EBCDIC character set.

The EBCDIC character set is described in section 10 of this chapter.

b) Other data formats

The internal representations of internal decimal, binary and floating-point (long or short) data formats are described under "USAGE Clause".

2. Alignment by insertion of slack bytes

There are two types of slack bytes: slack bytes within data records (intra-record slack bytes) and slack bytes between data records (inter-record slack bytes).

Intra-record slack bytes are unused character positions which precede every aligned data item in the record.

Inter-record slack bytes are unused character positions which are inserted between blocked logical data records.

a) Intra-record slack bytes

For an output file or in the WORKING-STORAGE section, the compiler inserts slack bytes within data records to ensure that all aligned data items are justified on the appropriate boundaries. For an input file or in the LINKAGE section, the compiler expects any required slack bytes to be present in order to ensure proper alignment of a data item declared as SYNCHRONIZED.

Since it is very important for the user to know the length of a data record in a file, the algorithm that the compiler uses to determine whether slack bytes are required and, if they are required, how many slack bytes are to be added, is described as follows:

The total number of occupied bytes in all elementary data items which precede a data item in a data record is computed, including any slack bytes previously added.

Example 1-7

This sum is to be divided by m, where:

m = 2 for COMPUTATIONAL data items with a length of 4 digits or less;

m = 4 for COMPUTATIONAL data items with a length of 5 digits or more;

m = 4 for COMPUTATIONAL-1 data items,

m = 8 for COMPUTATIONAL-2 data items,

m = 4 for index data items.

If the remainder r of this division is equal to zero, no slack bytes are required. If the remainder is unequal to zero, the number of slack bytes to be added is equal to m-r.

These slack bytes are added to each record immediately following the elementary item that precedes the COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2 or INDEX data item. They are declared as though they were a data item with a level number equal to that of the data item immediately preceding the aligned data item, and must be included in the size of the group where they are contained.

Example 1-8

Slack bytes within data records

```
01  A.
    02   B  PICTURE X(5).
    02   C.
         03 D  PICTURE XX.
        [03 slack byte PICTURE X. Inserted by the Compiler.]
         03 E  PICTURE S9(6) COMP SYNCHRONIZED.
```

Slack bytes are also added by the compiler when a group item is described with OCCURS clause and contains an aligned data item defined with USAGE as COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2 or INDEX. To decide whether to add slack bytes, the following steps are performed:

- The compiler calculates the size of the group including all intra-record slack bytes required.
- This sum is divided by the largest m required by any elementary item within the group.
- If the remainder r of this division is equal to zero, no slack bytes will be needed. If r is unequal to zero, m-r slack bytes must be added.

Insertion of slack bytes takes place at the end of each occurrence of the group item which contains the OCCURS clause, in order to ensure that all occurrences of table items begin at the same kind of boundary. In Example 9, all occurrences of D begin one byte beyond a double-word boundary.

Example 1-9

Occurrences of slack bytes in tables

```
01 A.
   02 B  PICTURE X.
   02 C  OCCURS 10 TIMES.
      03 D PICTURE X.
     [03 slack bytes PICTURE XX. Inserted by the compiler.]
      03 E PICTURE S9(4) V99 COMP SYNC.
      03 F PICTURE S9(4) COMP SYNC.
      03 G PICTURE X(5).
     [03 slack bytes PICTURE XX. Inserted by the compiler.]
```

If aligned data items defined as COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2 or INDEX follow an entry with an OCCURS DEPENDING clause, then slack bytes are added on the basis of the field which is repeated with the maximum number. If the length of this field is not divisible by the m required by the data, then only certain values of the data-name used in the DEPENDING phrase produce a correct alignment of the fields. The programmer should bear this situation in mind and try to avoid it. These values are ones in which the length of the data field, multiplied by the number of occurrences plus the number of slack bytes calculated on the basis of the maximum number of occurrences, is divisible by m with no remainder.

Example 1-10

Occurrences of slack bytes in a table with the DEPENDING phrase

```
01  A.
    02  B   PICTURE 99.
    02  C   PICTURE X OCCURS 50 TO 99 TIMES
             DEPENDING ON B.
   [02  slack bytes PICTURE X. Inserted by the compiler.]
    02  D   PICTURE S99 COMP SYNC.
```

In this example, when references to D are required, B is restricted to odd values.

```
01  A.
    02  B   PICTURE 999.
    02  C   PICTURE XX OCCURS 20 TO 99 TIMES
             DEPENDING ON B.
   [02  slack bytes PICTURE X. Inserted by the compiler.]
    05  D   PICTURE S99 COMP SYNC.
```

In this example, all values of B provide correct references to D.

b) Inter-record slack bytes

When data records that contain aligned data items are to be blocked, the programmer must ensure that all records following the first record in the input/output storage area are properly boundary-aligned.

The lengths of all items in the record, including all intra-record slack bytes, are added. Mode-V data records require that 4 bytes should be added for the record-length field. The total is then divided by the greatest m value of the items in that record.

If the remainder r of the division is zero, no slack bytes are needed between the records. If the remainder r is not equal to zero, m-r slack bytes will be needed. These slack bytes may be specified at the end of the record by writing a level-02 FILLER.

Example 1-11

Slack bytes between data records

The following example shows the method of calculating slack bytes both within and between data records. Consider the following data record description:

```
01  A.
    02  B   PICTURE X(5).
    02  C   PICTURE X(3).
    02  D   PICTURE X(3).
    02  E   PICTURE S9(4) COMP SYNC.
    02  F   PICTURE S9(5) COMP SYNC.
    02  G   PICTURE S9(4) COMP SYNC.
```

Altogether, the number of bytes in B, C and D is 11. E is a 4-digit data item with COMPUTATIONAL; therefore, a slack byte must be added to the record before E. Including this byte, the total number of bytes before F is thus 14. Since F is a 5-digit data item with COMPUTATIONAL, two slack bytes must be added beforehand. No slack bytes are required before G.

The revised record description is this:

```
01  A.
    02  B   PICTURE X(5).
    02  C   PICTURE X(3).
    02  D   PICTURE X(3).
   [02  slack byte PICTURE X. Inserted by the compiler.]
    02  E   PICTURE S9(4) COMP SYNC.
   [02  slack bytes PICTURE XX. Inserted by the compiler.]
    02  F   PICTURE S9(5) COMP SYNC.
    02  G   PICTURE S9(4) COMP SYNC.
```

As a result, the record which contains items with the COMP phrase has a total of 22 bytes. From the rules discussed earlier on, this yields m = 4 and r = 2.

Consequently, in order to achieve proper alignment for blocked data records, the user must add two slack bytes between records, at the end of each record.

Hence, the final record description is:

```
01  A.
    02  B   PICTURE X(5).
    02  C   PICTURE X(3).
    02  D   PICTURE X(3).
   [02  slack byte PICTURE X. Inserted by the compiler.]
    02  E   PICTURE S9(4) COMP SYNC.
   [02  slack bytes PICTURE XX. Inserted by the compiler.]
    02  F   PICTURE S9(5) COMP SYNC.
    02  G   PICTURE S9(4) COMP SYNC.
    02  FILLER PICTURE XX. [Inter-record slack bytes.]
```

1.4.2.6 Qualification

Function

Every user-specified name designating an element in the COBOL source program must be unique. A name is unique when there is no other name consisting of the same sequence of characters and hyphens, or the name occurs in a hierarchy of names, so that it can be referenced unambiguously. This occurs by specifying one or more names on a higher level of the hierarchy. The higher levels are called qualifiers, and the process that causes the name to be unique is called qualification. A name must be qualified sufficiently to be unique; however, it is not absolutely necessary to specify all levels of the hierarchy. Within the Data Division, all data names used for qualification purposes must be given a level number or a level identifier. Thus, two identical data names cannot be subordinate elements of a single group item, unless they can be uniquely qualified. In the Procedure Division, two identical paragraph names are not allowed to occur in the same section.

In the qualification hierarchy, the names belonging to a level identifier are the most important, followed by the names belonging to level 01, then those belonging to level 02, ... , 49. A section name is the only qualifier available for paragraph names. The uppermost name in the hierarchy must be unique, and cannot be qualified. Subscripted or indexed data names and conditional variables, as well as procedure names and data names, can be made unique by means of qualification. The name of a conditional variable can be used as a qualifier for each of its condition names.

Format 1

```
{data-name-1      }  [ [{OF}          ]  data-name-2 ]...
{condition-name-1 }    [ {IN}          ]
```

Format 2

```
paragraph-name  [ [{OF}          ]  section-name ]
                  [ {IN}          ]
```

Format 3

```
text-name  [ [{OF}          ]  library-name ]
             [ {IN}          ]
```

Rules

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

2. The same name must not appear on more than one level of the hierarchy.

3. A data name must not be subscripted or indexed when used as a qualifier.

Programming considerations

1. A data-name or a condition-name, if assigned to more than one data item within the source program, must be qualified whenever it is referenced in the Procedure, Environment or Data Division (except in the REDEFINES clause, where qualification is not needed and may be used).

2. A duplicate paragraph-name within a section is not allowable. When qualifying a paragraph-name by a section-name, the word SECTION must not be used. A paragraph-name, when referenced within the same section, need not be qualified.

3. A name may be qualified even when qualification is not required; if uniqueness may be ensured by more than one combination of qualifiers, then each such combination is permitted. The total set of the qualifiers for a given data-name must not be identical to a subset of qualifiers for another data-name.

4. Qualified data-names may have up to 49 qualifiers.

5. If more than one COBOL library is available to the compiler at object time, then every time text-name is referenced it must be qualified by library-name.

1.5 DECLARATIVES

Function

The Declaratives subdivision is an optional portion of the Procedure Division. It contains a group of procedures, called declarative procedures, which are not executed within the normal sequence of statements in the Procedure Division but only when a particular condition occurs.

Declarative procedures are used for performing the following functions:

input/output label handling
handling of input/output errors
special Report Writer functions.

Format (General format of the procedure division)

```
A        margin indication
|
|
|
PROCEDURE DIVISION [USING data-name-1 [data-name-2]...].
[DECLARATIVES.
{section-name SECTION. USE statement. [paragraph-name.[sentence]...]...}...
END DECLARATIVES.]
{section-name SECTION [segment-number].[paragraph-name.[sentence]...]...}...
```

Rules

1. Declarative procedures must be placed at the beginning of the Procedure Division, preceded by the keyword DECLARATIVES and followed by a period and a space. Declarative procedures are terminated by the keyword END DECLARATIVES, followed by a period and a space.

2. As mentioned in "General format of the procedure division", the Declaratives subdivision must be divided into sections. These sections are called declarative sections. Each declarative section contains a group of related procedures, and is preceded by a section header, immediately followed by a USE statement with subsequent period and space.

3. The USE statement defines the type of declarative procedures according to the three functions listed above. The formats of the USE statement are described in detail in chapter 3 ("Sequential File Organization": Formats 1 and 2), in chapters 4 to 6 ("Relative", "Indexed" or Direct File Organization": Format 2) and in chapter 8 ("Report Writer": USE BEFORE REPORTING Statement).

4. The USE statement itself is never executed; rather, it defines the conditions for executing the declarative procedures specified in the associated section.

1.6 STATEMENTS AND SENTENCES

There are three types of statements:

- conditional statements
- compiler-directing statements
- imperative statements.

These same three types also apply to sentences:

- conditional sentences
- compiler-directing sentences
- imperative sentences.

1.6.1 Conditional Statements and Conditional Sentences

1. A conditional statement says that the truth value of a condition is to be determined and that the resultant action in the object program is dependent upon this value.

   Conditional statements include:

   IF, SEARCH and RETURN statements.

   READ statements which contain the AT END or INVALID KEY phrase.

   WRITE statements which contain INVALID KEY or END-OF-PAGE phrase

   START, REWRITE or DELETE statements which contain the INVALID KEY phrase.

   ADD, COMPUTE, DIVIDE, MULTIPLY or SUBTRACT statements which contain the SIZE ERROR phrase.

   STRING and UNSTRING statements which contain the ON OVERFLOW phrase.

   GO TO statements which contain the DEPENDING ON phrase.

2. A conditional sentence is a conditional statement which may precede an imperative statement, and which is terminated by a period followed immediately by a space.

   Whenever an imperative statement occurs in the general format, it always refers to a set of imperative statements terminated either by a period, by an ELSE associated with an IF statement, or by a WHEN from a preceding SEARCH statement.

1.6.2 Compiler-Directing Statements and Compiler-Directing Sentences

1. A compiler-directing statement contains a compiler-related statement (i.e. USE or COPY) with appropriate operands.

   It causes the compiler to perform certain actions at compilation time.

Statements besides COPY which serve to make further use of the source program library can likewise be regarded as compile statements (see "COBOL User's Guide" [7]).

2. A compiler-directing sentence is a single compiler-directing statement which is terminated by a period followed immediately by a space.

1.6.3 Imperative Statements and Imperative Sentences

1. Any statement which is neither a conditional nor a compiler-directing statement is an imperative statement.

   An imperative statement causes a specific action to be carried out in the object program.

   It may consist of a set of imperative statements separated by COBOL separators.

ACCEPT	INITIATE	STOP
ADD (1)	INSPECT	STRING (3)
ALTER	MERGE	SUBTRACT (1)
CALL	MOVE	TERMINATE
CLOSE	MULTIPLY	UNSTRING (3)
COMPUTE (1)	OPEN	WRITE (6)
DELETE (2)	PERFORM (7)	
ENTRY	READ (5)	
DISPLAY	RELEASE	
DIVIDE (1)	REWRITE (2)	
EXIT	SET	
GENERATE	SORT	
GO TO (4)	START (2)	

   (1) without SIZE ERROR phrase
   (2) without INVALID KEY phrase
   (3) without ON OVERFLOW phrase
   (4) without DEPENDING ON phrase
   (5) without AT END or INVALID KEY phrase
   (6) without INVALID KEY or END-OF-PAGE phrase
   (7) without UNTIL phrase.

2. An imperative sentence is an imperative statement which is terminated by a period followed immediately by a space.

1.6.4 Categories of Statements

Arithmetic statements	ADD COMPUTE DIVIDE INSPECT (TALLYING) MULTIPLY SUBTRACT
Conditional statements	ADD (SIZE ERROR) COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DIVIDE (SIZE ERROR) GO TO (DEPENDING ON) IF MULTIPLY (SIZE ERROR) PERFORM (UNTIL) READ (END or INVALID KEY) RETURN (END) REWRITE (INVALID KEY) SEARCH (WHEN or AT END) START (INVALID KEY) STRING (OVERFLOW) SUBTRACT (SIZE ERROR) UNSTRING (OVERFLOW) WRITE (INVALID KEY or END-OF-PAGE
Data processing statements	ACCEPT (DATE, DAY, DAY-OF-WEEK or TIME) INSPECT CONVERTING INSPECT REPLACING MOVE SET (TO TRUE) STRING TRANSFORM UNSTRING
Input/output statements	ACCEPT (identifier) CLOSE DELETE DISPLAY OPEN READ REWRITE SET (ON/OFF) START STOP (literal) WRITE
End statement	STOP
Report Writer statements	GENERATE INITIATE TERMINATE

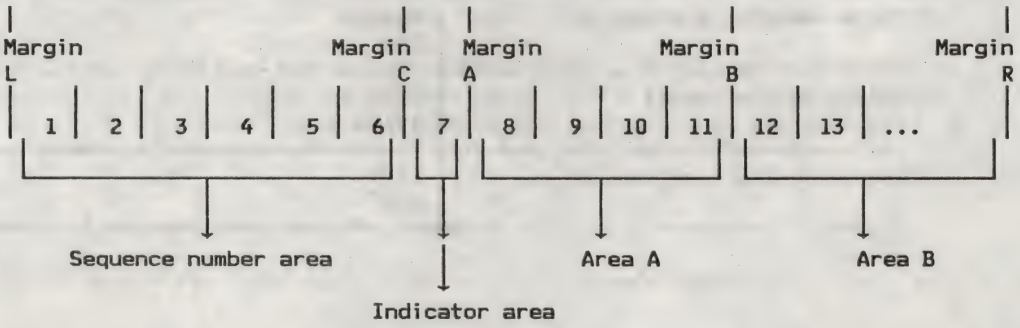
Program communication statements	CALL ENTRY EXIT PROGRAM
Procedure control statements	ALTER CALL EXIT GO TO PERFORM
Sort statements	MERGE RELEASE RETURN SORT
Table handling statements	SEARCH SET
Debugging statements	EXHIBIT TRACE
Compiler-directing statements	COPY USE

1.7 PROGRAM IMAGE FORMAT

1.7.1 General Description

The COBOL coding form provides a standardized method for writing down COBOL source programs in a standard program image format. This format is based in the 80 character positions in a line. Every compiler will accept COBOL source programs written in program image format and will generate a listing of the source program in the same format.

A line is divided as follows:



Margin L
is located to the left of the leftmost character position in a line.

Margin C
is located between the sixth and seventh character position in a line.

Margin A
is located between the seventh and eight character position in a line.

Margin B
is located between the eleventh and twelfth character positions in a line.

Margin R
is located to the right of the rightmost character position in a line.

Sequence number area
contains six character positions (columns 1 to 6) located between Margin L and Margin C.

Indicator area
the seventh character position in a line.

Area A
contains the character positions 8 through 11 and is located between Margin A and Margin B.

Area B
contains a finite number of character positions. It begins at the first character position to the right of Margin B and ends at the character position to the left of Margin R.

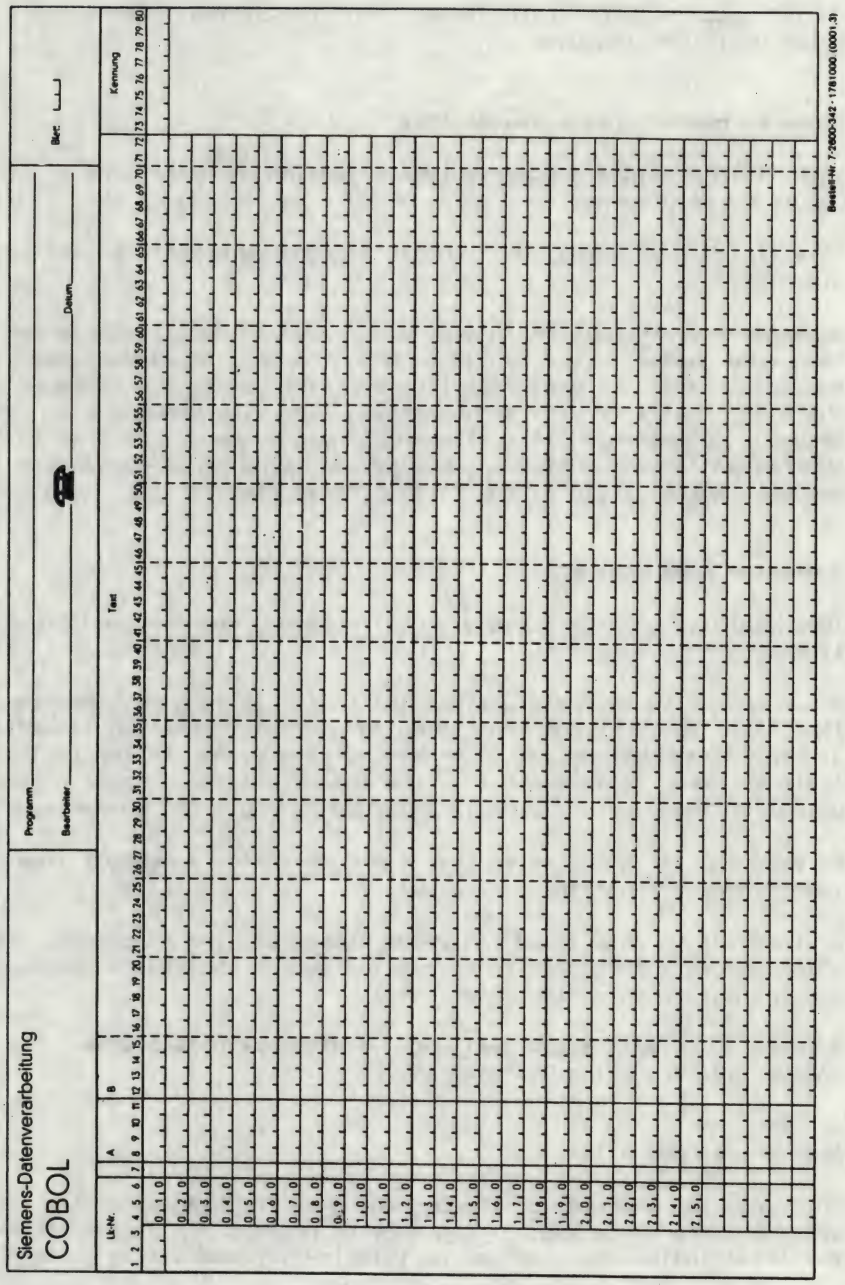


Figure 1-3 COBOL coding form

1.7.2 Rules for Using the COBOL Coding Form

COBOL programs are written out in a standardized format. Below you will find rules for using the COBOL coding form. These rules regarding the use of blanks take precedence over all other rules in this manual that govern the insertion or omission of blanks.

Figure 1-3 shows a COBOL coding form. Each line of the form has 80 character positions (columns).

1. **Sequence number area (columns 1-6)**

   This field provides a numeric identification of every line in a COBOL source program.

   This field, when used, must contain a sequence number consisting of six digits.

   Sequence numbers must be in ascending order. Otherwise, an error message (sequence number error) is issued when the program is compiled. If the sequence number area of a line contains spaces the sequence number check for that line will be suppressed. Checking resumes with the next sequence number which is available. Sequence number checking may be suppressed for the entire program by supplying space characters in the sequence number field of the first program line.

2. **Indicator area (column 7)**

   This field is used to designate continuation, comment, and diagnostic lines.

   A hyphen (-) in this field signifies that this is a **continuation line**, i.e. the previous line is being continued (see "Continuation of Lines") further below). The absence of a hyphen in the indicator field is taken to indicate that the last character in area B (see below) of the previous line is followed by one space character.

   An asterisk (*) supplied in this field indicates a **comment line** (see "Comment Line" further below).

   A slash (/) in this field indicates a special kind of comment line, which causes a form feed to be carried out in the source program listing before this line is printed.

   A letter D in this field designates a **diagnostic line** (see chapter 12 "Debugging Statements").

3. **Area A (columns 8-11)**

   This area is reserved for the beginning of the headers of the four COBOL program divisions, of the section headers and paragraph headings, for level indicators, and for certain level numbers (see Table 1-9).

4. **Area B (columns 12-72)**

   This area is the main field for entries of a COBOL source program. It is used to hold all those clauses and statements which do not have to begin in the A-area (see Table 1-9).

5. Program identification area (columns 73-80)

This area may be used to assign names to lines of a COBOL source program. It may contain any characters from the character set of the data processing system (EBCDIC) or may be blank. Its contents do not effect the compiler. It is advisable for reasons of clarity to provide parts of the program or the entire program with a meaningful name in this field.

6. Continuation of lines

A sentence or entry requiring more than one line may be continued on subsequent lines in area B. The first line is called a **continued line**, the following lines are called **continuation lines**. If a sentence or entry spans more than two lines then all lines, except the first and last, are both continued and continuation lines.

Two types of continuation are provided:

a) Continuation of nonnumeric literals

If a nonnumeric literal is continued on the next line(s), a hyphen should be entered in the indicator area (column 7) of the continuation line(s). The continuation may follow anywhere in area B (from column 13), immediately preceded by a quotation mark. All blanks located at the end of the continued line and/or following the quotation mark of the continuation line(s) are regarded as part of the literal.

b) Continuation of words and numeric literals

If a word or a numeric literal is continued on the next line, a hyphen must be entered in the indicator area (column 7) of the continuation line in order to show that the first non-blank character in area B of the continuation line is to immediately follow the last non-blank character of the continued line, i.e. without separating blank.

7. Blank lines

A blank line is a line that contains only spaces in columns 7 through 72. A blank line may appear anywhere in the source program, except immediately preceding a continuation line.

8. Program divisions, sections, paragraphs and description entries

Item	Convention for the placement of items
DIVISION header	Must be written starting at margin A (i.e. in column 8) and must appear on a line by itself.
SECTION header	Must be written starting at margin A; no other text except USE and COPY statements as well as segment numbers may appear on the same line.
Paragraph name	Must start at margin A.
Statements/clauses	Statements or clauses of a paragraph must be written within area B. The first sentence of a paragraph may begin on the line which contains the paragraph-name or on a new line.
Level indicators, file, sort-file and report description entry	The level indicators FD, SD and RD must be written starting at margin A; they must be followed, on the same line, starting at margin B, by the related file-name, sort-name or report-name and the associated explanatory information, if any.
Level numbers and file description entries	Level numbers 01 and 77 must be written starting at margin A; all other level numbers may begin anywhere in area A or B. The data description entries which are associated with a particular level number must begin in area B of the same line which contains that level number.

Table 1-9 COBOL margin conventions

9. Declaratives

The keywords DECLARATIVES and END DECLARATIVES which (respectively) open and close the declaratives section of the Procedure Division must each begin on a separate line. Both must be written starting at margin A, and both must be terminated by a period followed by a space.

10. Comment lines

Explanatory comments may be included anywhere in the COBOL source program in the form of comment lines by setting an asterisk or a slash in the indicator area (column 7). Any combination of characters from the character set of the data processing system (EBCDIC) may be used in areas A or B of these lines. The contents of the comment lines will be produced on the source program listing (at the top of a new page if a slash has been entered in the indicator area) and have no effect on the program.

11. Diagnostic lines

Diagnostic (debugging aid) lines may appear anywhere in the COBOL source program, following the OBJECT-COMPUTER paragraph. They are indicated by a "D" in the indicator area (see chapter 12, "Debugging Aids").

1.8 CREATING A COBOL PROGRAM

COBOL uses English words and certain grammatical rules which are derived from the English language. But COBOL is a programming language and demands absolute compliance with its rules. This is why the programmer will have to learn the COBOL rules described in the following sections of this manual.

The basic element of the COBOL language is the word, which may be for instance a reserved word or a user-defined word. Reserved words have a special syntactic significance and must be written exactly as shown in the list of the reserved words (Table 1-7). User-defined words are supplied by the programmer, e.g. for data-names and procedure-names. They must be formed in accordance with certain COBOL rules (see "Language Concepts, COBOL Words").

Reserved words and user-defined words are combined in a suitable manner by the programmer into clauses (in the Environment and Data Divisions) and statements (in the Procedure Division), subject to special format conventions. A clause or a statement describes only one action to be performed, one condition to be analyzed, or one data description. One or more clauses or one or more statements can be combined into one program sentence. Program sentences can be assembled to form paragraphs, which are named groups of logically linked sentences. Paragraphs can in turn be grouped together to form sections.

In the Procedure Division, paragraphs and sections may be referenced as procedures, and their names then generally referred to as procedure-names.

A COBOL language solution to a business data processing problem is formulated using four different elements:

1. Identification of the program.

2. Description of the equipment configuration and resources used.

3. Description of the data used.

4. Procedures to direct the processing of the data.

To allow these four elements to be specified separately, every COBOL program is structured into four divisions:

1. IDENTIFICATION DIVISION.

2. ENVIRONMENT DIVISION.

3. DATA DIVISION.

4. PROCEDURE DIVISION.

Table 2-1 shows, in detail, the general structure of a COBOL program. All further details are described in the appropriate sections of this manual. (See chapter 2.)

1.9 PROCESSING A COBOL PROGRAM

A COBOL **source program** must be supplied in a form which is acceptable to the data processing system; that is, the data and procedures described in a COBOL source program must be imaged to appropriate machine language representations of data and instructions so as to be immediately understandable to the data processing system. This conversion is achieved mainly by a **compiler**, which translates a source program into one or more **object modules**.

Executing the object code thus generated requires a **Linkage Editor**, which links an object module together with further compiler-generated object modules as well as runtime subroutines. The result is an **executable program** (object program).

The **runtime subroutines**, which are supplied as object modules, are used for performing special COBOL functions such as input/output operations.

Source programs and object programs are generally stored in a system library or a private library, where they are accessible to the various system utilities for program preparation as well as program tests.

The interactive debugging aids IDA and AID are available for hardware-oriented and symbolic tests of COBOL programs.

1.10 EBCDIC CHARACTER SET

Decimal	Hexadecimal	EBCDIC	Printer graphics
0	00	0000 0000	(LOW-VALUE)
.	.	.	
.	.	.	
.	.	.	
64	40	0100 0000	␣ (Space)
.	.	.	
.	.	.	
.	.	.	
74	4A	0100 1010	¢ (cents)
75	4B	0100 1011	. (period)
76	4C	0100 1100	< (less than)
77	4D	0100 1101	((left parenthesis)
78	4E	0100 1110	+ (plus)
79	4F	0100 1111	| (vertical)
80	50	0101 0000	& (ampersand)
.	.	.	
.	.	.	
.	.	.	
90	5A	0101 1010	! (exclamation mark)
91	5B	0101 1011	$ (dollar sign)
92	5C	0101 1100	* (asterisk)
93	5D	0101 1101	) (right parenthesis)
94	5E	0101 1110	; (semicolon)
95	5F	0101 1111	¬ (logical "NOT")
96	60	0110 0000	- (minus)
97	61	0110 0001	/ (slash)
.	.	.	
.	.	.	
.	.	.	
106	6A	0110 1010	¦ (logical AND)
107	6B	0110 1011	, (comma)
108	6C	0110 1100	% (percent)
109	6D	0110 1101	_ (underline)
110	6E	0110 1110	> (greater than)
111	6F	0110 1111	? (question mark)
.	.	.	
.	.	.	
.	.	.	
122	7A	0111 1010	: (colon)
123	7B	0111 1011	# (number sign)
124	7C	0111 1100	@ (at the rate of)
125	7D	0111 1101	' (apostrophe)
126	7E	0111 1110	= (equals)
127	7F	0111 1111	" (quote)
.	.	.	
.	.	.	
.	.	.	
193	C1	1100 0001	A
194	C2	1100 0010	B
195	C3	1100 0011	C
196	C4	1100 0100	D
197	C5	1100 0101	E
198	C6	1100 0110	F

Decimal	Hexadecimal	EBCDIC	Printer graphics
199	C7	1100 0111	G
200	C8	1100 1000	H
201	C9	1100 1001	I
.	.	.	
.	.	.	
.	.	.	
209	D1	1101 0001	J
210	D2	1101 0010	K
211	D3	1101 0011	L
212	D4	1101 0100	M
213	D5	1101 0101	N
214	D6	1101 0110	O
215	D7	1101 0111	P
216	D8	1101 1000	Q
217	D9	1101 1001	R
.	.	.	
.	.	.	
.	.	.	
226	E2	1110 0010	S
227	E3	1110 0011	T
228	E4	1110 0100	U
229	E5	1110 0101	V
230	E6	1110 0110	W
231	E7	1110 0111	X
232	E8	1110 1000	Y
233	E9	1110 1001	Z
.	.	.	
.	.	.	
.	.	.	
240	F0	1111 0000	0
241	F1	1111 0001	1
242	F2	1111 0010	2
243	F3	1111 0011	3
244	F4	1111 0100	4
245	F5	1111 0101	5
246	F6	1111 0110	6
247	F7	1111 0111	7
248	F8	1111 1000	8
249	F9	1111 1001	9
.	.	.	
.	.	.	
.	.	.	
255	FF	1111 1111	◇ (lozenge) (HIGH-VALUE)

2 BASIC ELEMENTS OF A COBOL PROGRAM

2.1 INTRODUCTION

A COBOL language solution to a business data processing problem is formulated using four different elements:

1. Identification of the program.
2. Description of the equipment configuration and resources used.
3. Description of the data used.
4. Procedures to direct the processing of the data.

To allow these four elements to be specified separately, every COBOL program is structured into four divisions:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

Table 2-1 shows, in detail, the general structure of a COBOL program. All further details are described in the appropriate sections of this manual.

```
[IDENTIFICATION DIVISION]
 ID DIVISION.

 PROGRAM-ID. program-name.

[AUTHOR.          [comment-entry]...]

[INSTALLATION.    [comment-entry]...]

[DATE-WRITTEN.    [comment-entry]...]

[DATE-COMPILED.   [comment-entry]...]

[SECURITY.        [comment-entry]...]

[  ENVIRONMENT DIVISION.
 [  CONFIGURATION SECTION.
  [SOURCE-COMPUTER. [entry.]]

  [OBJECT-COMPUTER. [entry.]]

  [SPECIAL-NAMES.   [entry.]]
 [  INPUT-OUTPUT SECTION.
  [FILE-CONTROL.    [entry.]...]

  [I-O-CONTROL.     [entry.]...]]

[   DATA DIVISION.
 [  FILE SECTION.
  [ file description entry.          {record description entry. }...]...
                                     {                          }
  [ sort-file description entry.     {record description entry. }...]...

 [  WORKING-STORAGE SECTION.
  [{77-level description entry. }]...
  [{                            }]
  [{record description entry.   }]
  [{                            }]

  [ LINKAGE SECTION.              ]
  [{77-level description entry. }]...
  [{                            }]
  [{record description entry.   }]

[   REPORT SECTION.
 [  report description entry.  {report group description entry. }...]...

 [  SUB-SCHEMA SECTION.
 [data base description entry. ]

[   PROCEDURE DIVISION [USING data-name-1 [data-name-2]...].
 [  DECLARATIVES.
 {  section-name SECTION.USE statement.[paragraph-name.[sentence]...]...}...
    END DECLARATIVES.

[  {section-name SECTION [segment-number].[paragraph-name.[sentence]...]...}... ]
```

Table 2-1 General structure of a COBOL source program

Although a COBOL source program may have an unlimited number of lines, the compiler will only number these lines uniquely up to a value of 32767.

2.2 IDENTIFICATION DIVISION

2.2.1 General Description

The Identification Division is the first part of every COBOL program and identifies the program by specifying the program-name.

In addition the user may include the following: the date the program was written, the date the program is compiled, and other descriptive information explaining the purpose of the program.

2.2.2 Structure

General Format

```
A       margin indication
|
|
V
[IDENTIFICATION DIVISION.]
{                        }
[ID DIVISION.            ]

 PROGRAM-ID.  program-name.
[AUTHOR.            [comment-entry]...]

[INSTALLATION.      [comment-entry]...]

[DATE-WRITTEN.      [comment-entry]...]

[DATE-COMPILED.     [comment-entry]...]

[SECURITY.          [comment-entry]...]
```

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION or ID DIVISION, followed by a period. This subheading must be followed by the PROGRAM-ID paragraph, which sets down the name of the program.

All paragraphs following the PROGRAM-ID paragraph are optional. In Standard COBOL, any of these paragraphs included in the COBOL program must be presented in the order shown. However, the compiler described herein will accept these paragraphs in any order.

Every Identification Division paragraph except the PROGRAM-ID paragraph contains comment entries. A comment entry can be any combination of characters from the EBCDIC character set. The continuation of comment lines using a hyphen in the indicator field is prohibited; however, the comment entry may extend over two or more lines.

The paragraphs with the names AUTHOR, INSTALLATION, DATE-WRITTEN and SECURITY, serve only documentation purposes of the user and are neither evaluated nor modified by the compiler.

PROGRAM-ID PARAGRAPH

Function

The PROGRAM-ID paragraph provides the name by which a program is identified. This paragraph must be present in every program.

Format

```
PROGRAM-ID. program-name.
```

Rules

1. The program-name may be 1 to 30 characters in length. It must consist of letters and digits and must begin with a letter.

2. The operating system uses only the first eight characters of program-name for identifying the module. Therefore, these characters should be unique for every name in a particular module/program library.

3. The program-name identifies the source program and all reports associated with it. It also identifies the compiled object program for the purpose of program communications.

DATE-COMPILED PARAGRAPH

Function

The DATE-COMPILED paragraph inserts the compilation date on the source program listing.

Format

```
DATE-COMPILED. [comment-entry]...
```

Rules

1. A comment-entry may consist of any combination of characters from the EBCDIC set, organized to conform to sentence and paragraph structure. A continuation of comment-entries by using the hyphen in the indicator field is not permitted; the comment-entry may, however, span more than one line.

2. If the DATE-COMPILED paragraph is specified, the current date is written on the listing. The whole comment-entry on the first line of this paragraph is replaced with the current date. Any comment-entries on subsequent lines within the DATE-COMPILED paragraph are left intact.

2.3 ENVIRONMENT DIVISION

2.3.1 General Description

The Environment Division provides a standard method for describing those aspects of a data processing problem which depend on the physical characteristics of a given computer installation. This division can be used to define the equipment configuration of the data processing system on which the program is to be compiled and executed. It also provides an opportunity for specifying input/output control, specific machine characteristics, and control techniques.

The Environment Division must be present in every COBOL source program.

The Environment Division consists of two sections:

1. CONFIGURATION SECTION
2. INPUT-OUTPUT SECTION.

The Configuration Section deals with the special features of the computers used for compiling and executing the program.

This section is divided into three paragraphs:

1. the SOURCE-COMPUTER paragraph, which describes the equipment configuration of the data processing system on which the source program is to be compiled

2. the OBJECT-COMPUTER paragraph, which describes the equipment configuration of the data processing system on which the object program generated by the compiler is to be executed

3. the SPECIAL-NAMES paragraph, which relates the implementor names used by the compiler to the mnemonic names used in the source program.

The INPUT-OUTPUT SECTION deals with the required information for controlling the transmission and handling of data between external devices and the object program.

This section is divided into two paragraphs:

1. the FILE-CONTROL paragraph, which names the files and assigns them to external devices

2. the I-O-CONTROL paragraph, which describes special control techniques to be used in the object program.

The following types of **file organization** are available to the COBOL user:

- sequential
- indexed
- relative

The way in which logical data records are created and retrieved depends on the existing data organization (where "organization" means "data structure on a physical file"). There can be no compatibility between any two different types of file organization. The organization of an input file must be the same as at the creation time of that file.

The following access methods are available to the COBOL user:

- sequential
- random
- dynamic/extended

Sequential access:

Reading and writing data records in sequential order. At file creation time the data records are placed in succession on that file. When reading the file, the data records are located in the order in which they appear in the file.

Random access:

Reading and writing data records in nonsequential order. Every data record on a random-access file is provided with a key. Access to a data record requires that its key be supplied.

Dynamic/extended access:

Switching between sequential and random access method.
Extended access can only be specified for files with indexed organization, the dynamic access method only for files with relative or indexed organization.

The access methods permitted for each type of file organization are indicated in Table 2-2 and discussed in chapters 3 to 6.

Access to indexed and relative files presupposes the use of clauses in the FILE-CONTROL paragraph:

- Access to indexed files is made by using the RECORD KEY clause.
- Access to relative files is made by using the RELATIVE KEY clause.

The description entries of these clauses contain additional rules for file access and describe the processes which run in the COBOL object program for sequential, random and extended/dynamic access.

For further details of file processing terminology, see those sections in chapters 3 to 5 in which file organization is discussed.

File organization and possible actions	Permissible access methods			
	Sequential	Random	Extended	Dynamic
Sequential files				
Record creation	X			
Record retrieval	X			
Record update	X *			
Relative files				
Record creation	X	X		X
Record retrieval	X	X		X
Record update	X	X		X
Record addition		X		X
Indexed files				
Record creation	X		X	X
Record retrieval	X	X	X	X
Record update	X *	X	X	X
Record addition	X	X	X	X

Table 2-2 Valid access methods for the different types of file organization

*) In sequential access, updating a data record means updating in place, i.e. a READ immediately followed by a REWRITE. This is allowable for mass-storage files only.

2.3.2 Structure

General format

```
A      margin indication
|
|
ENVIRONMENT DIVISION.

[ CONFIGURATION SECTION.
[SOURCE-COMPUTER.      [source-computer-entry][debugging-mode].]
[OBJECT-COMPUTER.      [object-computer-entry.]]
[SPECIAL-NAMES.        [special-names-entry.]]

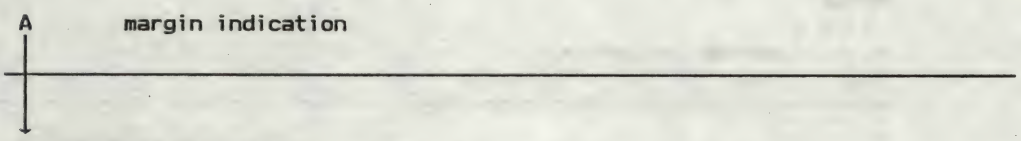
[ INPUT-OUTPUT SECTION.
[FILE-CONTROL.         [file-control-entry.]]
[I-O-CONTROL.          [input-output-control-entry.]]
```

2.3.3 Configuration Section

Function

The Configuration Section indicates the computer installation on which the program is to be compiled and executed and, optionally, relates system-names to user-supplied mnemonic names.

Format




```
CONFIGURATION SECTION.

[SOURCE-COMPUTER.        [source-computer-entry][debugging-mode].]

[OBJECT-COMPUTER.        [object-computer-entry.]]

[SPECIAL-NAMES.          [special-names-entry.]]
```

Rules

1. All sections and paragraphs must be written starting at margin A.

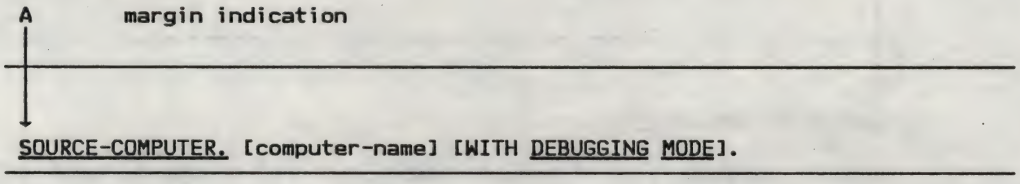
2. The CONFIGURATION SECTION and its associated paragraphs, with the exception of the SPECIAL-NAMES paragraph, are required in every COBOL source program. The compiler, however, allows the SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs to be omitted and will treat the whole CONFIGURATION SECTION as optional.

SOURCE-COMPUTER PARAGRAPH

Function

The SOURCE-COMPUTER paragraph identifies the data processing system on which the source program is to be compiled; it may also be used to specify debugging aids.

Format



Programming consideration

This paragraph serves for documentation only, unless the DEBUGGING MODE clause is supplied.

2

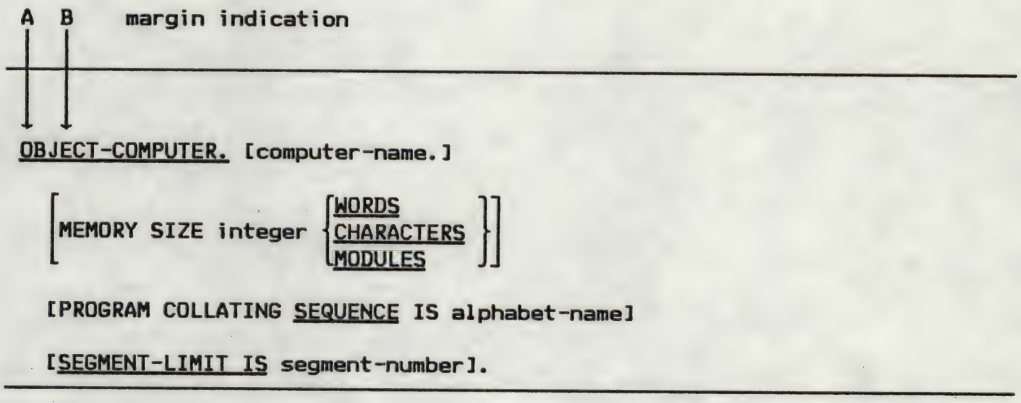
OBJECT-COMPUTER PARAGRAPH

Function

The OBJECT-COMPUTER paragraph describes the data processing system on which the program is to be executed.

Additional information is provided on COBOL program segmentation (see chapter 9 "Segmentation").

Format



Rules

1. computer-name as well as the MEMORY-SIZE clause are used for documentation purposes only and are treated as comments.

2. computer-name is a system-name, and consists of the term SIEMENS-7000 and the model number, i.e. SIEMENS-7.755.

3. If the PROGRAM COLLATING SEQUENCE clause is specified, the collating sequence associated with alphabet-name (see "SPECIAL-NAMES Paragraph") is used to determine the truth value of any nonnumeric comparisons:

   a) explicitly specified in "Relation Conditions".
   b) explicitly specified in "Condition-Name Condition".
   c) implicitly specified by the presence of a CONTROL clause in a report description entry (see chapter 8 "CONTROL Clause").

4. If the PROGRAM COLLATING SEQUENCE clause is not specified, the native collating sequence is used (EBCDIC).

5. If the PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with the alphabet-name specified in that clause.

6. The PROGRAM COLLATING SEQUENCE clause is also applied to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE phrase of the respective MERGE or SORT statement is specified (see chapter 10, "MERGE Statement", "SORT Statement").

7. The PROGRAM COLLATING SEQUENCE clause applies only to the program in which it is specified.

8. segment-number may be any integer between 0 and 49 (see chapter 9, "Segmentation").

9. The effect of the SEGMENT LIMIT clause is discussed (in chapter 9, "Segmentation").

For sample PROGRAM COLLATING SEQUENCE and ALPHABET clauses see "SPECIAL-NAMES Paragraph".

SPECIAL-NAMES PARAGRAPH

Function

The SPECIAL-NAMES paragraph serves the following purposes:

1. It relates system-names to mnemonic-names specified by the user.
2. It relates alpha-names to character code sets and/or collating sequences.
3. It defines symbolic characters.
4. It relates class-names to a character set defined by the user.
5. It supplies a character which is to be used in place of the currency sign ($) in picture-strings.
6. It exchanges the functions of the comma and the decimal point in picture-strings as well as in numeric literals.

Format

```
A B    margin indication
|  |
|  |
SPECIAL-NAMES.

[ implementor-name { [IS mnemonic [ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2]]]
                   [IS mnemonic [OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]]] }...
                   { [ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2]]
                   [OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]] }... ]

[ ALPHABET alphabet-name-1 IS { [STANDARD-1]
                              [STANDARD-2]
                              [NATIVE]
                              [literal-1 [{THROUGH}literal-2]]
                                         [{THRU   }          ]
                                         [ALSO literal-3 [ALSO literal-4]...]
                              [literal-5 [{THROUGH}literal-6]]...
                                         [{THRU   }          ]
                                         [ALSO literal-7 [ALSO literal-8]...] } ]... ]

[ SYMBOLIC CHARACTERS { {symbolic-character-1}... [IS ] {integer-1}... }... [IN alphabet-name-2] }... ]
                                                  [ARE]

[ CLASS class-name IS { literal-9 [{THROUGH}literal-10] }... ]...
                      [{THRU   }               ]

[CURRENCY SIGN IS literal-11]

[DECIMAL-POINT IS COMMA].
```

Rules

1. implementor-name is a system-name and must be a name from the left-hand column of Table 2-3 below.

   Implementor-names and their meanings:

Implementor-name	Meaning
CONSOLE	Control console or subconsole
TERMINAL or SYSOUT	Remote terminal at which the user enters his tasks
SYSIN, SYSIPT or SYSRDR	System logical input file
SYSLST SYSLST01-SYSLST99	System logical output file
SYSPUNCH, SYSPCH or SYSOPT	System logical punch file
CSP, C01 etc.	For printer carriage control (see Table 2.4)
JV-job-variable-name	Job variable describing the link name of a job variable (see below)
literal	Single-character nonnumeric literal identifying Report Writer output (see below)
TSW-0 to TSW-31	Task switches
USW-0 to USW-31	User switches
COMPILER-INFO	Compiler information
CPU-TIME, PROCESS-INFO, TERMINAL-INFO	Operating system information

Table 2-3 Meaning of implementor-names

When specified, the implementor-names CSP, C01 to C08, C10 or C11 represent the actions described in Table 2-4 below:

Implementor-name	Meaning
CSP	Suppress spacing
C01 to C08	Skip to channel 1 through 8
C10 to C11	Skip to channel 10 or 11

Table 2-4 Printer carriage control by implementor-name

job-variable-name indicates a BS2000 job variable. It is a COBOL word of up to 7 characters, and must not begin or end with a hyphen. It is used to form the link name *name and for accessing the job variable.

literal defines the character to be used when identifying the printer lines belonging to a report. The mnemonic-name assigned to the literal must also appear in the CODE clause of the report description entry (see "Report Description (RD)").

One single-character nonnumeric literal of this sort may be specified for each report defined in the source program.

2. If the literal phrase is specified in the ALPHABET clause, any given character for literal-1, literal-2 etc. which is referenced by alphabet-name in the PROGRAM COLLATING SEQUENCE clause (see "OBJECT-COMPUTER paragraph") or in the COLLATING SEQUENCE phrase of the SORT or MERGE statement (q.v.) may be used once only.

3. The following rules apply to literal-1, ..., literal-11:

   a) If the literals are numeric, they must be unsigned integers with a value from 1 to 256.

   b) If the literals are nonnumeric and are connected with the THROUGH, THRU or ALSO phrase, each literal must be one character long.

   c) The literals may not be the figurative constant symbolic-character. literal-11 must not be a figurative constant.

4. The words THROUGH and THRU are equivalent.

5. The following rules apply to symbolic-character:

   a) No given character may be used more than once in the SYMBOLIC CHARACTERS clause.

   b) The relationship between each separate symbolic-character and its corresponding integer results from the sequence within the SYMBOLIC CHARACTERS clause: symbolic-character-1 is connected with integer-1, symbolic-character-2 with integer-2 and so forth.

   c) The number and sequence of the symbolic characters must be exactly identical with the number and sequence of integers.

   d) The position specified within the collating sequence by integer-1 must exist in the hardware-specific character set. If the IN phrase is specified, the position within the collating sequence must exist in the character set named by alphabet-name-2.

   e) The internal representation of symbolic-character is identical to that of the corresponding character in the hardware-specific character set.

   f) symbolic-character is a figurative constant.

6. The CLASS clause enables the user to associate a name with the character set defined in this clause. This class-name may be referenced in the class condition only. Characters specified as the values of literal-9, literal-10, ... form a special character set named by class-name.

   The value of each literal indicates:

   a) The ordinal number of a character in the hardware-specific character set when the literal is numeric.

   b) The currently valid character itself when the literal is nonnumeric. When the value of this nonnumeric literal contains more than one character, each of these characters belongs to the character set named by the class-name.

   If the THROUGH phrase is specified, the character sequence beginning with literal-9 and ending with literal-10 forms the special character set indicated by class-name within the hardware-specific character set. The THROUGH phrase can be used to specify this character sequence in either ascending or descending order.

7. When used in the CURRENCY SIGN clause, literal-11 is limited to a single character and must not be one of the following:

   1. digits 0 to 9
   2. alphabetic characters A, B, C, D, E, L, P, R, S, V, X, Z, or the space
   3. special characters
      * (asterisk)
      + (plus)
      - (minus)
      , (comma)
      . (period)
      ; (semicolon)
      ((left parenthesis)
) (right parenthesis)
      " (quotation mark)
      / (slash)
      = (equal sign)

8. Unless the CURRENCY SIGN clause is written, only the $ can be used as the currency symbol in a picture-string.

9. The DECIMAL-POINT IS COMMA clause is used to exchange the functions of period and comma in numeric literals and in picture-strings.

Programming considerations

1. Even when two or more clauses are specified in the SPECIAL-NAMES paragraph, only one period, following the last clause, is allowed.

2. Note regarding the sequence of declarations:
   DECIMAL-POINT IS COMMA, when used, must be the last clause specified; the sequence of the remaining declarations is arbitrary.

3. If implementor-name is not a user or task switch, the associated mnemonic-names may be used in ACCEPT, DISPLAY and WRITE statements.

4. If implementor-name is a user or task switch, at least one condition-name must be associated with it. The status of the switch is described under "Condition-Names", and can be queried by testing the condition-name (see "Switch-Status Conditions").

   The status of a switch may be modified by using a SET statement, format 3 (see "SET Statement").

   Example 2-1

   Use of job variables:

```
ID DIVISION.
PROGRAM-ID. JVTEST.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    JV-JV1 IS JOB-VAR-1.
         .
         .
         .
```

   Prior to the program call:

```
/DCL JV1TEST,LINK=*JV1
```

5. When C01 to C08, C10, C11 or CSP is specified as implementor-name, the associated mnemonic-name can be used in a WRITE clause with ADVANCING phrase.

6. The ALPHABET clause provides a means for relating a name to a particular character code set and/or collating sequence. When referenced in the PROGRAM COLLATING SEQUENCE clause (see "OBJECT-COMPUTER Paragraph") or the COLLATING SEQUENCE phrase of a SORT or MERGE statement (see "SORT Statement" or "MERGE Statement"), alphabet-name specifies a collating sequence. The character code type is defined by referencing alphabet-name-1 in the SYMBOLIC-CHARACTERS clause or the CODE-SET clause of a file description entry (for sequentially organized files).

   a) If the STANDARD-1 phrase is specified, the character code set or collating sequence is that defined in American National Standard Code for Information Exchange (ASCII), X3.4-1968.

   b) If the STANDARD-2 phrase is specified, the character code type indicated is the International Reference Version of the ISO 7-bit Code, as defined in International Standard 646, "7-Bit Coded Character Set for Information Processing Interchange". Each character of the Standard character set is associated with a

corresponding character from the native character set.

c) If the NATIVE phrase is specified, the native character code set or native collating sequence is used (EBCDIC).

d) If the literal phrase of the ALPHABET clause is specified, the alphabet-name may not be referenced in a CODE-SET clause (see "CODE-SET Clause").

- Example 2-2

  CLASS DIGIT1 IS "0" "1".

  The character set of digits 0 and 1 is associated with the class-name DIGIT1. This name can be used in a class condition (cf. 2-80) to check whether the contents of a data field consist of a combination of the digits 0 and 1.

- The value of the literal specifies the ordinal number of a character within the native character set, if the literal is numeric.

  Example 2-3

```
 65 indicates space
 78 indicates (
194 indicates A
242 indicates 1
```

  This value must not exceed the value which represents the number of characters in the native character set (256).

- The value of the literal specifies the actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

  Example 2-4

```
ALPHABET ALPHATAB IS "AJKCDF".

First character is A
Second character is J
      .
      .
      .
Sixth character is F
```

- The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

  Example 2-5

```
ALPHABET ALPHATAB IS "A" "C" "D" "Z".

First character in the collating sequence is "A"
Second character in the collating sequence is "C"
Third character in the collating sequence is "D"
Fourth character in the collating sequence is "Z"
```

- Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.

Example 2-6

```
ALPHABET ALPHATAB IS "ABC" "D" "Z".
```

The lowest ordinal positions in the collating sequence receive, respectively,

1. A
2. B
3. C
4. D
5. Z

All remaining non-specified characters, e.g. L, M, and N, receive higher ordinal positions,

where:
L has a lower ordinal position than M,
M has a lower ordinal position than N.

- If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified.

  In addition, the set of contiguous characters specified by a given THROUGH phrase may contain characters of the native character set in either ascending or descending sequence.

Example 2-7

```
ALPHABET ALPHATAB IS "A" THRU "I".
```

In the native character set the characters lying between A and I are B, C, D, E, F, G, and H. Hence the collating sequence is sorted in ascending order:

```
First character is A
Second character is B
Third character is C
      .
      .
      .
Eighth character is H
Ninth character is I
```

Example 2-8

```
ALPHABET ALPHATAB IS "F" THRU "A".
```

In the native character set, the characters lying between F and A are E, D, C, and B. Hence, the collating sequence is sorted in descending order:

```
First character is F
Second character is E
      .
      .
      .
Fifth character is B
Sixth character is A
```

- If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1, literal-3, literal-4, ..., are assigned to the same position in the collating sequence being specified.

Example 2-9

```
ALPHABET ALPHATAB IS "A" ALSO "B" ALSO "C" ALSO "D".
```

The characters A, B, C, and D will be assigned to the lowest ordinal positions in the collating sequence.

7. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position in the program collating sequence, the last character specified is assigned the figurative constant HIGH-VALUE.

Example 2-10

```
ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 194.
```

The highest ordinal position in the collating sequence is occupied by the character which appears at the 194th position of the native character set, i.e. the character A.

A is assigned to the figurative constant HIGH-VALUE:

Example 2-11

```
ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 197, "A" ALSO "B" ALSO "C".
```

Positions 1 through 193 of the collating sequence are assigned the characters which appear at positions 193 to 1 of the native character set.

Positions 194 through 253 of the collating sequence are assigned the characters which appear at positions 255 to 197 of the native character set.

Position 254 is assigned the characters A, B, C; with this all characters in the native character set have been assigned a position in the collating sequence. The highest-order position (254) is assigned the characters A, B, C. Being the character specified last, C is assigned to the figurative constant HIGH-VALUE.

8. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position in the program collating sequence, the first character specified is assigned the figurative constant LOW-VALUE.

Example 2-12

```
ALPHABET ALPHATAB IS "0" "1" "2".
```

The lowest ordinal character in the collating sequence is 0. Hence 0 is assigned to the figurative constant LOW-VALUE.

Example 2-13

```
ALPHABET ALPHATAB IS "A" ALSO "B" ALSO "C".
```

The lowest ordinal position in the collating sequence is assigned the characters A, B, C. The character A, which was specified first, is assigned to the figurative constant LOW-VALUE.

9. The DECIMAL-POINT IS COMMA clause is used to exchange the functions of the decimal point and the comma in numeric literals and in picture-strings. When this clause is used, the decimal point required in a numeric literal or in a picture-string must be represented by a comma. The decimal point, in turn, must be used for the function normally performed by the comma.

Example 2-14

PROGRAM COLLATING SEQUENCE and ALPHABET Clause:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER.
    PROGRAM COLLATING SEQUENCE IS ALPHATAB.
SPECIAL-NAMES.
    ALPHABET ALPHATAB IS "X" "Y" "Z".
DATA DIVISION.
WORKING-STORAGE SECTION.
77  ITEM-1 PIC X(3) VALUE "ABC".
77  ITEM-2 PIC X(3) VALUE "XYZ".
PROCEDURE DIVISION.
    IF ITEM-1 > ITEM-2...
```

With the definition of the alphabet-name ALPHATAB in the SPECIAL-NAMES paragraph the character X was assigned to the first position in the collating sequence, Y to the second and Z to the third.

All remaining characters of the native character set are assigned a position in the collating sequence implicitly, since their positions in the collating sequence are higher than that of the specified characters X, Y, Z and their order in the collating sequence was taken from the native character set without alteration.

Positions 1 through 231 in the native character set correspond to positions 4 through 234 in the collating sequence.

Positions 235 through 256 in the native character set correspond to positions 235 through 256 in the collating sequence.

Thus, A occupies position 197, B position 198, and C position 199.

Hence, the relation ITEM-1 > ITEM-2 is **true**.

Example 2-15

PROGRAM COLLATING SEQUENCE and ALPHABET Clause:

```
ID DIVISION.
PROGRAM-ID.ALPH.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER.
    PROGRAM COLLATING SEQUENCE IS ALPHA.
SPECIAL-NAMES.
    ALPHABET ALPHA 1 THRU 247, 251 THRU 256
                   "7" ALSO "8" ALSO "9".
DATA DIVISION.
WORKING-STORAGE SECTION.
77  ITEM-1 PIC X(3) VALUE HIGH-VALUE.
77  ITEM-2 PIC X(3) VALUE "789".
PROCEDURE DIVISION.
    IF ITEM-1 = ITEM-2 DISPLAY "FIRST RELATION OK" UPON TERMINAL
    ELSE DISPLAY "FIRST RELATION NOT OK" UPON TERMINAL.
    IF ITEM-2 = HIGH-VALUE DISPLAY "SECOND RELATION OK"
    UPON TERMINAL
    ELSE DISPLAY "SECOND RELATION NOT OK" UPON TERMINAL.
    STOP RUN.
```

Characters less than 7 remain as in native collating sequence.
Characters greater than 9 are then appended, thereby becoming less than 7.
The characters 7, 8, 9 are set at the highest ordinal position, with 9, being the last character specified, corresponding to "HIGH-VALUE".

Result:

```
First relation ok
Second relation ok
```

2.3.4 INPUT-OUTPUT Section

Function

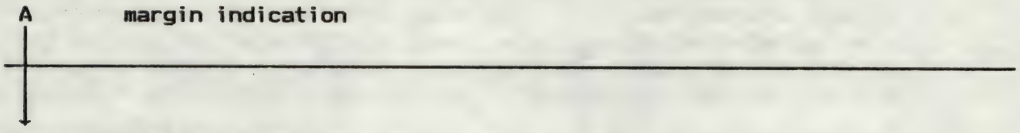
The INPUT-OUTPUT Section covers the definition of each file, names its external devices, assigns each file to one or more input/output devices, and also provides the information needed for an efficient transmission of data between the input/output devices and the object program.

This section is divided into two paragraphs:

the FILE-CONTROL paragraph, which names the files used in the program and assigns each file to external devices

the I-O-CONTROL paragraph, which indicates special input/output techniques.

Format




```
INPUT-OUTPUT SECTION.

[FILE-CONTROL.   [file-control-entry.]...]

[I-O-CONTROL.    [input-output-control-entry.]]
```

Rules

1. All sections and paragraphs must begin at margin A.

2. The entire INPUT-OUTPUT Section is optional.

The paragraphs of the INPUT-OUTPUT Section are dealt with in the following chapters:

Chapter 3 "Sequential File Organization"

Chapter 4 "Relative File Organization"

Chapter 5 "Indexed File Organization"

2.4 DATA DIVISION

2.4.1 General Description

Two types of data are processed by a COBOL program:

1. data stored on some external medium

2. data generated internally during program execution.

The first type of data is combined into records and files; the second type of data must be declared by the user as data records or subordinate data items.

To ensure maximum independence of data from its specific representation on external volumes and in data processing systems, the properties or contents of data are described with respect to a standard data format rather than a system-oriented format. This format is adapted to the general application of data processing and uses decimal numbers for representing numbers and the rest of the COBOL character set for representing nonnumeric characters.

The data description method used allows a distinction to be made between the physical and system-dependent properties on the one hand, and conceptual characteristics on the other.

Physical and certain system-specific properties of data on an external medium are defined in a COBOL source program in order that efficient use may be made of special techniques.

1. The term "physical properties of data" refers to

   a) the grouping of logical data records within the physical boundaries of the external volume

   b) the recording mode in which the data is stored on the external volume.

2. The term "system-dependent properties of data" refers to

   the description under which a file is identified on some external medium.

Most of these properties are described in chapters 3, 4 and 5 under the different types of file organization.

The conceptual characteristics of data on an external volume pertain to the logical units of data, the logical data records, and are not associated with any physical or system-specific properties.

These characteristics are described:

for files: in chapters 3, 4 and 5;

for records: in the Data Division record description entries.

A data description in a COBOL program is separate from the declaration of execution procedures. This permits the programmer a great number of options for modifying a data description entry without any change to the procedures which are related to that entry. Therefore, to a certain extent, the procedures of a COBOL program may be seen as data-independent.

2.4.2 Structure

The Data Division is one of the mandatory divisions of a source program. It is divided into five sections as follows:

1. FILE SECTION,
   see chapters 3 to 5, "File Organization"

2. WORKING-STORAGE SECTION

3. LINKAGE SECTION,
   see chapter 8, "Program Communication"

4. REPORT SECTION,
   see chapter 9, "Report Writer"

5. SUB-SCHEMA SECTION
   (For further information see the "UDS Reference Manual" [6]).

All data which is, or is meant to be, stored on external media must first be described in the **FILE SECTION** before it can be processed by a COBOL program.

Information intended for internal use must be described in the **WORKING-STORAGE SECTION**:

Information transmitted from one program to another must be described in the **LINKAGE SECTION.**

The contents and appearance of all listings created by the Report Writer must be described in the **REPORT SECTION.**

All information pertaining to the description of data base structures must be entered in the SUB-SCHEMA SECTION. (For further details see the "UDS Reference Manual" [6]).

The following pages present the general format for the sections of the Data Division and define the order in which they should be entered in the source program.

General format

```
  A      margin indication
  |
  |
  V
  DATA DIVISION.

[ FILE SECTION.                                                          ]
[file description entry.      {record description entry.}...]...        ]
[                             {                          }              ]
[sort-file description entry. {record description entry.}...]...        ]

[ WORKING-STORAGE SECTION.                                               ]
[ {[77-level description entry.]}...                                     ]
[ {                            }                                         ]
[ {[record description entry.  ]}                                        ]

[ LINKAGE SECTION.                                                       ]
[ {[77-level description entry.]}...                                     ]
[ {                            }                                         ]
[ {[record description entry.  ]}                                        ]

[ REPORT SECTION.                                                        ]
[report description entry.{report group description entry.}...]...       ]

[ SUB-SCHEMA SECTION.                                                    ]
[data-base description entry. ]
```

2.4.2.1 General Rules

1. The Data Division is specified by, and must begin with, the division header DATA DIVISION.

2. The File Section is specified by, and must begin with, the section header FILE SECTION. This entry is followed by one or more file description entries and one or more data record description entries.

3. The Working-Storage Section is specified by, and must begin with, the section header WORKING-STORAGE SECTION. This entry is followed by 77-level entries for noncontiguous items and/or record description entries in any order.

4. The Linkage Section is specified by, and must begin with, the section header LINKAGE SECTION. This entry is followed by 77-level entries for noncontiguous items and/or record description entries in any order.

5. The Report Section is specified by, and must begin with, the section header REPORT SECTION. This entry is followed by one or more report description entries and one or more report group description entries.

6. The SUB-SCHEMA SECTION is specified by, and must begin with, the section header SUB-SCHEMA SECTION. This entry is followed by the entry for the sub-schema library which is to be processed by the compiler.

7. Each section of the Data Division is optional and may be omitted from the source program when unnecessary. However, when used, the sections must appear in the order stated in "General Format".

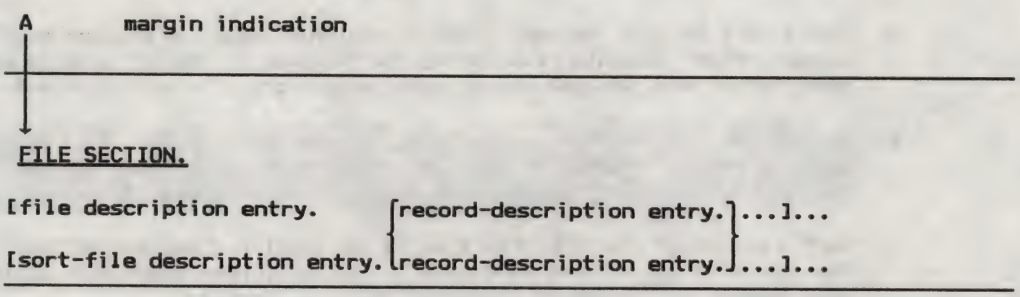
8. For proper indentation of the division and section headers, refer to chapter 1, "COBOL Coding Form".

2.4.2.2 FILE SECTION

Function

The FILE SECTION defines how files are set up. Each file is defined by a file description entry or a sort-file description entry and one or more record description entries, or it is defined by a file description entry and one or more report description entries (see also under "General Format"). Record description entries are written immediately following the file description entry. When a file description entry defines a file which is to be used as an output file for the Report Writer, the only record description entries permitted for this file are those associated with the report file. Report description entries are supplied in the REPORT SECTION of the Data Division.

Format



Rules

On this, also refer to "General Rules" of the Date Division. File description entries are explained under "Data Description Entries for Group Items within a Record Description Entry".

For further details see chapters 3 to 5 and chapter 10.

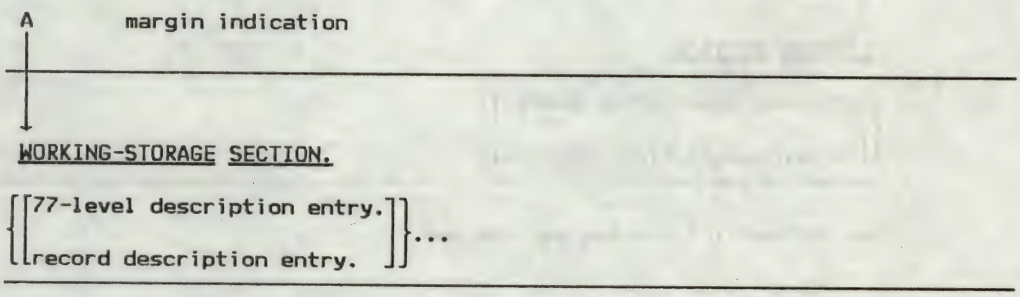
2.4.2.3 WORKING-STORAGE SECTION

Function

The WORKING-STORAGE SECTION describes data records and structurally noncontiguous data items (refer to 'General Format') which are not part of external files and whose values are generated and used internally.

They also describe data items whose values are assigned in the source program and which will not be modified when the object program is executed.

Format

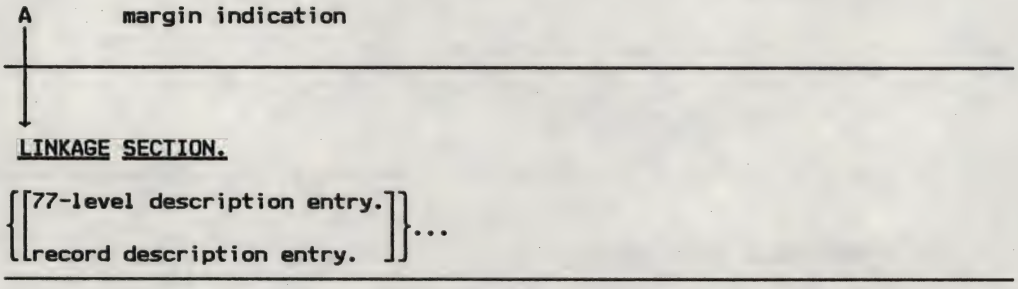


2.4.2.4 LINKAGE SECTION

Function

The LINKAGE SECTION is located in a called program and describes data items in the calling program, which may itself reference the called program.

Format



For further information see chapter 7.

2.4.2.5 REPORT SECTION

Function

The REPORT SECTION describes the contents and formats of the listings which are to be produced.

Format

```
REPORT SECTION.

[report description entry. {report group description entry}...]...
```

Programming considerations

1. The REPORT SECTION must be the last section in the Data Division.
2. The report description entry is discussed in the following pages.
3. All report group description entries must follow the related report description entry in a body.
4. A report group entry formally corresponds to a record description entry of the FILE or WORKING-STORAGE sections. The report group entry comprises all of the data description entries for a single report group. The first entry in the report group description begins with level number 01. All associated data description entries begin with level number 02 (see "Report Group Description").

For further information see chapter 8.

2.4.3 Organization of the Entries of the Data Division

As shown in the "General Format", each section of the Data Division consists of one or more entries. Each Data Division entry begins with a **level indicator** or a **level number**, followed by one or more spaces, followed by the name of a data item (except in the SUB-SCHEMA and REPORT sections), followed by a sequence of independent clauses describing the data item. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level number.

1. Level indicators

Table 2-5 shows the level indicators and the corresponding entries in the Data Division, whose beginning is marked by a level indicator.

Level indicator	Use	Description
FD	File description	See chapters 3 to 5, "File Description"
SD	Sort-file description	See chapter 10 "Sort-File Description"
RD	Report description	See chapter 8 "Report Description"
DB	Data-base description	See "UDS Reference Manual" [6].

Table 2-5 Summary of level indicators

A report description (RD) entry must be provided for each report, in addition to the file description (FD) entry for the file on which the report is to be generated (see chapter 8, "Report Writer").

Each level indicator must be written starting at margin A and is followed from margin B by its associated data-name and corresponding descriptive information.

2. Level numbers

Table 2-6 shows the allowable level numbers and their associated Data Division entries.

Level-number	Use
01	Record description entries or report group description entries
02 - 49	Data description entries describing subdivisions of a record or of a report group
77	Description entries for independent or noncontiguous data items which are not subdivisions of other items and are not themselves subdivided
66	Elementary items or group items described by the RENAMES clause for the purpose of regrouping data items (see "RENAMES Clause")
88	Condition-name entries to specify condition-names associated with particular values of a conditional variable (see "VALUE Clause")

Table 2-6 Summary of level numbers

Level numbers are used in structuring a logical data record so that subdivisions of the record may be referenced. Once a record has been subdivided, this structuring may be carried further to permit even more detailed data references.

The level numbers 01 and 77 must be written starting at margin A, followed by their associated data names and appropriate descriptive information. All other level numbers may begin either at margin A or margin B, followed, from margin B, by associated data names and appropriate descriptive information.

A single-digit level number is either written as a space character followed by a digit, or as a zero followed by a digit. There must be at least one space character to separate the level number from the item which follows the level number. There must be at least one space between the level number and the word that follows.

Consecutive data description entries may have the same format as the first such entry or may be intended according to their level numbers. While indentation is helpful for documentation purposes, it does not affect the compiler.

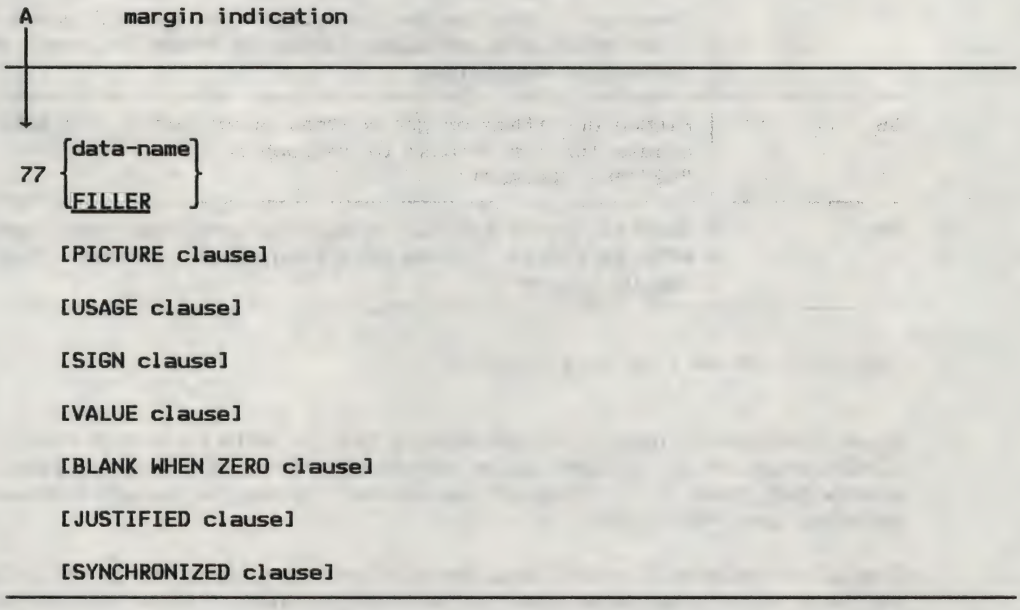
For further rules, refer to 'Data Description Clauses', 'Level Numbers'.

2.4.4 77-Level Entries in the WORKING-STORAGE and LINKAGE Sections

Function

Description entries at the 77-level describe data items in the WORKING STORAGE or LINKAGE sections which bear no hierarchical relationship to one another and which need not be further subdivided. Therefore, they do not have to be arranged in data records. Instead, they are classified and declared as independent or structurally noncontiguous elementary items.

Format



Rules

At least one of the following two clauses must be specified in a 77-level data description:

1. PICTURE clause
2. USAGE clause with INDEX, COMPUTATIONAL-1 or COMPUTATIONAL-2 phrase.

The clauses may be written in any order. In the following discussion, they appear in alphabetical order.

2.4.5 Data Description Entry

2.4.5.1 General Description

Data description entry is the general term for the description of every single data item in the Data Division; the entry is composed of the level number, followed by a data-name (if necessary) and several data clauses.

It should be noted here that record and report group description entries are used in different ways:

The **record description entry** is used to define all data description entries which are associated with a particular record; that is, the record description entry describes all properties of that record.

Multiply-defined 01- and 77-level record description entries are not treated as errors provided they are not used in the Procedure Division.

The concept of level is contained in the structure of a logical data record. This concept arises from the need of assigning names to the parts of a record in order to access them. Once a record has been thus subdivided, the subdivision can be carried further to permit even more detailed data references.

A "report group" is to the REPORT SECTION what a "data record" is to other sections of the Data Division. The **report group description entry** describes all data description entries associated with a particular report group. Within a report group description entry, a distinction is made between the first and the subsequent data description entries (see chapter 8, "Report Writer").

Those components of a data record which are not further subdivided are called **elementary items**; a record thus either consists of a sequence of elementary items or is itself an elementary item.

An elementary item may be at the most 65535 bytes long.

In order to reference a number of elementary items at one time, these items are arranged into **"groups"** or **"group items"**. These groups may in turn be arranged into sets of two or more groups. Consequently, an elementary item may belong to more than one group.

The word **"data item"** is used in those cases where there is no need to distinguish between elementary and group items.

Data description entries may be classified according to the clauses which are either permitted or required in their formats:

1. 77-level entries in the WORKING-STORAGE or LINKAGE section. These entries describe elementary items which are not part of a record description entry and have no subdivisions.

2. Data description entries for group items within a record description in the FILE, WORKING-STORAGE or LINKAGE sections.

   These entries describe group items which have at least one subordinate elementary item each and are themselves subordinate to a record description entry.

3. Data description entries for elementary items within a record description in the FILE, WORKING-STORAGE or LINKAGE sections.

   These entries describe elementary items which are part of a record description entry but have not subdivisions.

Points 1 to 3 are discussed in greater detail below.

4. A 66-level entry in the FILE, WORKING-STORAGE or LINKAGE sections.

   A data description entry of this sort defines the regrouping of data items, each of which is described by a data entry. It is not applicable to data description entries of the levels 01, 77 or 88.

   This entry is described in the RENAMES clause.

5. An 88-level entry in the WORKING-STORAGE, LINKAGE, or REPORT sections.

   This sort of entry is used to define a condition-name.
   It is described in format 2 of the VALUE clause.

6. A data description entry for the first entry in a report group description.

   This entry describes 01-level data items of a given report group.

7. A data description entry for all entries in a report group description other than the first.

   This entry describes report group data items with level numbers other than level number 01.

Items 6 and 7 will be discussed more fully in chapter 8, "Report Writer".

2.4.5.2 Level Numbers

Function

A level number shows the hierarchy of data within a logical record (see chapter 1, "Glossary"). It also qualifies entries for data items within the WORKING-STORAGE and LINKAGE sections, as well as for condition-names and data-items in the RENAMES clause.

Format

```
level-number
```

Rules

1. The level number is a special numeric literal consisting of one to two digits. A level number which is less than 10 may be written either as a single digit or with a leading zero.

2. Level numbers 01 and 77 must be entered starting at margin A. All other level numbers may begin in area A or area B. Only in the case of 01- and 77-level entries does this effect compilation.

3. File description or sort-file description entries must have level numbers with values of 01 to 49, 66, or 88.

4. Record description entries are allowed level numbers 01 and 02 only.

5. Data description entries in the WORKING-STORAGE or LINKAGE section must have level numbers with values of 01 to 49, 77, 66, or 88.

6. Level number 01 qualifies the first entry of each record description or report group description.

7. Special level numbers are assigned to certain kinds of entries for which the level concept does not really apply. These numbers include:

   Level number 66 is used to qualify renaming entries. It may be used only as described in the RENAMES clause.

   Level number 77 is used to qualify structurally noncontiguous data items in the WORKING-STORAGE and LINKAGE sections. It may be used only as described under "77-level Description Entry".

   Level number 88 refers to entries which define condition-names assigned to a condition variable. It may be used only as described in format 2 of the VALUE clause.

Programming considerations

1. In every data description entry the first element must be a level number.

2. Multiple 01-level entries which are subordinate to a given level indicator (except RD) implicitly represent redefinitions of the same area.

3. For reasons of documentation it is generally very helpful to indent level numbers (see Rule 2). For this reason, most data record descriptions are written in indented form.

4. It is not possible to define edited items at level number 01 within a file description entry.

Example 2-16

```
01  ADDRESS.
    02  NAME.
        03  FIRST-NAME              PIC X(18).
        03  LAST NAME               PIC X(20).
    02  STREET-ADDRESS.
        03  STREET                  PIC X(16).
        03  HOUSE-NUMBER            PIC 9(4).
        03  CITY                    PIC X(15).
        03  STATE                   PIC XX.
        03  ZIP-CODE.
            04  DIGIT-1             PIC 9.
            04  DIGIT-2             PIC 9.
            04  DIGIT-3             PIC 9.
            04  DIGIT-4             PIC 9.
            04  DIGIT-5             PIC 9.
            04  DIGIT-6             PIC 9.
```

The statement

```
MOVE ADDRESS TO...
```

will move the entire group.

The statement

```
MOVE NAME TO...
```

will move the first and last names.

Etc.

2

2.4.5.3 Data Description Entries for Group Items within a Record Description

in the FILE, WORKING-STORAGE and LINKAGE Sections

Function

Description of data items within a record which are subdivided by subordinate items.

A data description entry of this sort is part of the record description and is followed by at least one data description entry (within the same record description) which is subordinate to the described data entry.

An 01-level entry is of this kind if a subordinate 02-level entry is specified.

Format

```
A       margin indication
|
|
v
level-number  {data-name}
              {FILLER   }

              [REDEFINES clause]

              [USAGE clause]

              [SIGN clause]

              [VALUE clause]

              [SYNCHRONIZED clause]

              [OCCURS clause].
```

Rule

The clauses may be written in any order, except for the FILLER and REDEFINES clauses which, when used, must be written as shown in the format. In the following discussion, the clauses appear in alphabetical order.

2.4.5.4 Data Description Entries for Elementary Items within a Record

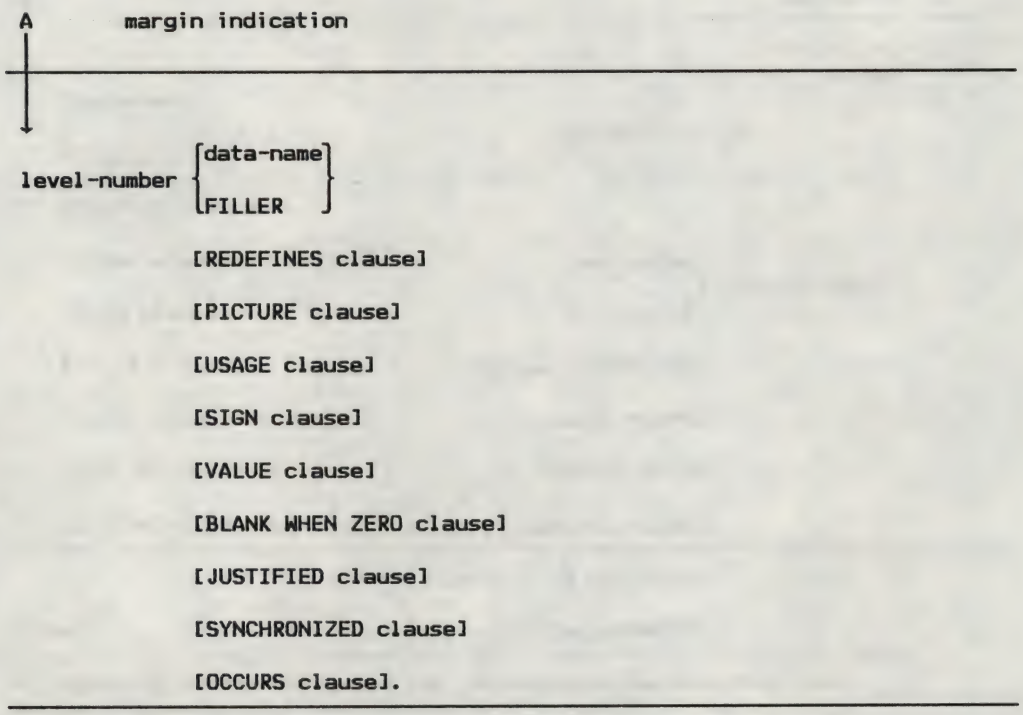
Description in the FILE, WORKING-STORAGE and LINKAGE Sections

Function

Description of data items within a record which are not further subdivided by subordinate items.

Any 01-level entry is of this type if not followed by associated entries of level numbers 02 to 49.

Format



Rules

At least one of the following two clauses must be specified in a data description entry:

1. PICTURE clause
2. USAGE clause with INDEX, COMPUTATIONAL-1 or COMPUTATIONAL-2 phrase.

The clauses may be written in any order, except for the REDEFINES clause which, when used, must be written as shown in the format. In the following discussion, the clauses are described in alphabetical order.

BLANK WHEN ZERO CLAUSE

Function

The BLANK WHEN ZERO clause specifies that an item is to be set to blanks whenever its value is zero.

Format

BLANK WHEN ZERO

Rules

1. When the BLANK WHEN ZERO clause is used, the item will contain only blanks if the value of the item is zero.

2. When the BLANK WHEN ZERO clause is used for numeric data items, the category of the item is considered to be numeric-edited.

Programming considerations

1. The BLANK WHEN ZERO clause may be specified only at the elementary level for numeric-edited or numeric items.

2. The BLANK WHEN ZERO clause may not be used for data items of level number 66 or 88.

3. If the BLANK WHEN ZERO clause and the PICTURE clause with asterisk (*) (for zero suppression) are used simultaneously in a data description entry, the zero suppression editing function overrides the function of the BLANK WHEN ZERO clause (see "PICTURE Clause").

Example 2-17

Description:

```
02 PURCHASE PICTURE $Z.99,BLANK WHEN ZERO.
```

Value of AMOUNT: 0

Statement: MOVE AMOUNT TO PURCHASE.

Value of PURCHASE after the MOVE statement:

␣␣␣␣␣(5 spaces)

DATA-NAME OR FILLER CLAUSE

Function

A data-name specifies the data being described. The reserved word FILLER specifies an elementary or group item which is never referenced and, therefore, need not be given a name.

Format

```
level-number [data-name]
             [FILLER   ]
```

(The level number is not part of the data-name or FILLER clause; it is shown here only for clarity.)

Rules

1. data-name must be formed according to the rules for user-defined words.

2. A data-name is a name defined by the user to specify a data item in a program. A data-name refers to the type of data rather than to a particular value; the data item that is referenced by data-name may receive different values during the execution of the program.

3. The reserved word FILLER is used to give a name to an elementary item or group item which is never referenced in the program, and therefore does not required a data-name. A FILLER data item cannot be referenced directly. FILLER data items are ignored in a MOVE, ADD or SUBTRACT statement with CORRESPONDING phrase (see "CORRESPONDING Phrase"). If the data-name or FILLER entry is omitted, FILLER is assumed.

Programming considerations

1. All 01- and 77-level entries in the WORKING-STORAGE and LINKAGE sections must be given unique data-names, since neither type of entry can be qualified. A subordinate data-name need not be spelled uniquely, if it can be qualified in a manner which makes it unique.

2. In the FILE, WORKING-STORAGE and LINKAGE sections, the first word in a data description entry following the level number must be a data-name or the reserved word FILLER.

Example 2-18

```
01 CARD-RECORD.
   02 FIRST-NUMBER        PICTURE 9(8).
   02 SECOND-NUMBER       PICTURE 9(12).
   02 FILLER              PICTURE X(60).
```

Here, a card is identified by the data-name CARD-RECORD, and its first two fields are identified by the data-names FIRST-NUMBER and SECOND-NUMBER. Since the third field is not referenced in the program, its level number is followed by the reserved word FILLER.

2

JUSTIFIED CLAUSE

Function

The JUSTIFIED clause is used for the alignment of nonnumeric data in a nonnumeric receiving item in a manner not conforming to the standard.

Format

```
[JUSTIFIED]
{      }  RIGHT
[JUST     ]
```

Rules

1. JUST is the abbreviation of JUSTIFIED.

2. If the JUSTIFIED clause is specified for the receiving item, and the sending item is longer than the receiving item, justification is made on the rightmost character position, and the leftmost character of the source item are truncated.

   If the JUSTIFIED clause is specified for the receiving item, and the receiving item is longer than the sending item, the data is justified on the rightmost character position; the remaining unused character positions on the left of the receiving item are padded with blanks.

3. When the JUSTIFIED clause is omitted, the rules for data alignment within an elementary item are applicable (see under "Alignment").

Programming considerations

1. The JUSTIFIED clause can be specified for elementary items only.

2. The JUSTIFIED clause can must not be specified for numeric or edited data items.

3. The JUSTIFIED clause must not be specified for data items of level number 66 or 88.

Example 2-19

Normal positioning:

Sending item

J	O	E

Receiving item

J	O	E		

Positioning when JUSTIFIED clause is specified:

J	O	E

		J	O	E

OCCURS CLAUSE

Function

The OCCURS clause shows the number of elements contained in a table, and supplies information on the use of subscripts or indices. This clause is described in chapter 6, "Table Handling".

2

PICTURE CLAUSE

Function

The Picture clause describes the general characteristics and editing requirements of an elementary data item.

Format

```
{PICTURE}
{       } IS picture-string
{PIC    }
```

Rules

1. PIC is the abbreviation of PICTURE.

2. A picture-string consists of certain allowable combinations of the characters in the COBOL set, used as symbols. These combinations determine the class of data to which an item belongs.

3. There are 18 characters or symbols that may be used in a picture-string: A, comma (,), X, 9, P, Z, *, B, 0, +, minus (-), currency symbol ($), slash (/), S, V, period (.), credit (CR), and debit (DB). The functions of each character and symbol are described under "PICTURE-String Character and Symbol Summary" (see further below).

4. The characters S, V, ., CR, and DB may appear only once in a PICTURE clause.

5. An integer enclosed in parentheses can follow the symbols A, comma (,), X, 9, P, Z, *, $, B, slash (/), 0, minus (-), and plus (+), to indicate the number of consecutive occurrences of the symbol. The number in parentheses must not exceed 65535.

6. The allowable combinations of symbols used in a picture-string are shown in Table 2-7.

   An X at an intersection means that, in a picture-string, the "second symbol" specified in the header line of the associated column may be located at any position to the **right** of the "first symbol" located at the start of the line.

7. At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, -, or CS 1) must be present in a picture-string.

8. Nonfloating insertion symbols + and -, floating insertion symbols Z, *, +, -, and CS 1), and the symbol P appear twice in Table 2-7. The leftmost column and uppermost row for each symbol represent its use to the left of the decimal point.

1) CS is the abbreviation for "currency symbol".

First symbol			Second symbol: Non-floating insertion symbols									Second symbol: Floating insertion symbols							Second symbol: Other symbols				
			B	0	/	.	.	+ −	+ −	CR DB	CS1)	Z *	Z *	+ −	+ −	CS1)	CS1)	9	A X	S	V	P	P
								l	r			l	r	l	r	l	r					l	r
Non-float. insertion symbols	B		X	X	X	X	X		X	X		X	X	X	X	X	X	X	X		X	X	
	0		X	X	X	X	X		X	X		X	X	X	X	X	X	X	X		X	X	
	/		X	X	X	X	X		X	X		X	X	X	X	X	X	X	X		X	X	
	,		X	X	X	X	X		X	X		X	X	X	X	X	X	X			X	X	
	.		X	X	X	X			X	X			X		X		X	X					
	+ or −	l	X	X	X	X	X				X	X	X			X	X	X			X	X	X
	+ or −	r																					
	CR or DB																						
	CS1)		X	X	X	X	X		X	X		X	X	X	X			X			X	X	X
Float. insertion symbols	Z or *	l	X	X	X	X	X		X	X		X	X					X			X	X	
	Z or *	r	X	X	X	X			X	X			X										
	+ or −	l	X	X	X	X	X							X	X			X			X	X	
	+ or −	r	X	X	X	X									X								
	CS1)	l	X													X	X	X			X	X	
	CS1)	r	X	X	X	X			X	X							X						
Other symbols	9		X	X	X	X	X		X	X								X	X		X	X	
	A or X		X	X	X													X	X				
	S																	X			X	X	X
	V		X	X	X	X			X	X			X		X		X	X					X
	P	l							X	X											X	X	
	P	r	X	X	X	X			X	X			X		X		X	X					X

Table 2-7 Precedence of symbols used in the PICTURE clause
l = left of the decimal point
r = right of the decimal point

1) CS is the abbreviation for the currency symbol.

9. The maximum number of characters that may be written in a picture-string is 30. This does not limit the number of characters in the represented area, which may be much more than 30.

10. The number of characters specified in the picture-string is used to determine the size of the item. However, the actual core storage requirements are determined by the combination of the PICTURE and USAGE clauses (see "USAGE Clause").

11. Five data categories may be described with the PICTURE clause:

    alphabetic
    alphanumeric
    numeric
    alphanumeric edited
    numeric edited

12. There are two general methods of performing editing in the PICTURE clause: by insertion, or by suppression and replacement.

    Four types of insertion editing are available:

    simple insertion
    special insertion
    fixed insertion
    floating insertion

    Two types of suppression and replacement editing are available:

    zero suppression and replacement with spaces
    zero suppression and replacement with asterisks (*).

Programming considerations

1. The PICTURE clause is permitted only for elementary items.

2. The picture CLAUSE must be specified for all elementary items, except for index data items and internal floating-point data items.

3. **Picture-string character and symbol summary**

   The characters and symbols that are permitted in a picture-string have the following meaning:

   A Each A in the picture-string represents a character position that may contain only a letter of the alphabet or a space.

   B Each B in the picture-string represents a character position into which a space character will be inserted. Each space is counted in the size of the item.

   P P is the scaling character. It represents a numeric digit position for which, however, storage is never reserved and which is always treated as if it contained a zero. P (or a group of Ps) indicates the location of an assumed decimal point (to the left of the Ps if the leftmost characters of the picture-string are the Ps, or to the right of the Ps if the Ps are the rightmost characters of the picture-string). The maximum number of digit positions in the scaling character for numeric and numeric-edited items is 30.

   The character V (see below) may be specified or omitted. When specified, it must be inserted in the position of the assumed decimal point, to the left or to the right of the P or Ps specified. The scaling character P is not counted in the size of the data item. But the scaling characters are counted in determining the maximum number of digit positions (30) in numeric-edited or numeric data items. The scaling character P and the insertion character . (period) must not be specified at the same time in a picture-string.

   S The character S indicates the presence but not the location or mode of representation of an operational sign. If used, it must be the leftmost character of the picture-string. The S is not counted in the size of the item unless it is part of a SIGN clause which contains the SEPARATE CHARACTER phrase.

V The character V indicates the position of an assumed decimal point. Since a numeric item cannot contain a printed decimal point, an assumed decimal point simply provides the compiler with information about the scaling alignment of items involved in computations. Storage is never reserved for the character V; therefore, V is not counted in the size of the item. If the assumed decimal point is the rightmost character in the picture-string, the character V need not be supplied.

X Each X in the picture-string represents a character position which may contain any character from the EBCDIC set.

Z Each Z in the picture-string represents a leading numeric character position. If such a character position contains a zero, then the zero is replaced by a space. Each Z is counted in the size of the item.

9 Each 9 in the picture-string represents a character position that contains a numeral and is counted in the size of the data item.

0 Each zero in the picture-string represents a character position into which the numeral zero will be inserted. Each zero is counted in the size of the item.

/ Each slash (/) in the picture-string represents a character position into which a slash will be inserted. Each slash is counted in the size of the item.

, Each comma (,) in the picture-string represents a character position into which a comma is inserted. Each comma is counted in the size of the data item.

. The period (.) in the picture-string is an editing symbol and represents the decimal point on which the data item is aligned. Additionally, it represents a character position into which a period is inserted. The period is counted in the size of the data item.

Note

The functions of the period and the comma may be interchanged in a program when the DECIMAL-POINT IS COMMA clause was specified in the SPECIAL-NAME paragraph of the Environment Division. The rules governing the decimal point are then applicable to the comma, and vice versa, whenever they appear in a picture-string.

\+
\-
CR
DB

These symbols are used as editing sign control symbols. When used, each represents the character position into which the editing sign control will be placed. These symbols are mutually exclusive in any one picture-string, and each character used in the symbol is counted in determining the size of the data item.

Editing sign control symbols produce different results for positive and negative data items, depending on the value (see "Rules for Fixed Insertion Editing").

* This symbol is a check protection symbol. Each asterisk (*) in the picture-string represents a leading numeric character position into which an asterisk will be placed if that position contains a zero. Each asterisk (*) is counted in the size of the item.

$ The currency symbol ($) in the picture-string represents a character position into which a currency sign is to be placed. The currency symbol in a picture-string is represented either by the symbol $ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

4. **Alphabetic data items**

Rules

a) In the picture-string of an alphabetic item, only the characters A and B are allowed.

b) Any combination of the 26 letters of the alphabet and the space from the COBOL character set may be specified as the contents of a data item.

Example 2-20

Picture	Value
PICTURE AAA	NEW
PICTURE BAAAB	-NEW-

5. **Alphanumeric data items**

Rules

a) The picture-string for an alphanumeric data item may contain certain combinations of the following symbols: A, X, 9.

An alphanumeric item is treated as if its picture-string contained all X's, with each X representing one character position.

b) A picture-string that contains all A's or 9's does not define an alphanumeric item.

Example 2-21

The alphanumeric value AB1234 could be represented by any of the following picture-strings:

```
PICTURE XXXXXX
PICTURE AAXXXX
PICTURE AA9999
PICTURE A(2)X(4)
```

6. **Numeric data items**

There are two types of numeric data items: fixed-point items and floating-point items.

Fixed-point items

There are three types of fixed-point items: external decimal, binary, and internal decimal (see "USAGE Clause").

Rules

a) The picture-string for a fixed-point data item may contain any valid combination of the following symbols: 9, V, P, S.

b) It can contain from one up to and including 18 digit positions.

c) If the symbol S has not been specified, a data item may contain a combination of the digits 0 through 9.

d) If the symbol S has been specified, the data item may contain, in addition to the above digits, a "+", "-" or other representation of an arithmetic sign (see "SIGN Clause").

Example 2-22

Valid combinations for fixed-point items:

```
PICTURE 9999
PICTURE S99
PICTURE S99V9
PICTURE PPP999
PICTURE S999PPP
```

Example 2-23

Let 8735 be the contents of a data item in the following examples.

For PICTURE P(4)9(4), the arithmetic value of this field is .00008735
For PICTURE 9(4)P(2), the arithmetic value of this field is 873500.
The arithmetic value is used for all operations except DISPLAY.

Floating-point data items

There are two types of floating point data items:
internal and external floating-point items (see "USAGE Clause").

Rules

a) The picture-string for an external floating-point item has the following format:

$$\left\{\begin{matrix}+\\-\end{matrix}\right\} \text{mantissa E} \left\{\begin{matrix}+\\-\end{matrix}\right\} \text{exponent}$$

where the following rules are to be observed for each element of the picture-string:

$\left\{\begin{matrix}+\\-\end{matrix}\right\}$ A positive or negative sign must be written immediately preceding the mantissa and the exponent in the picture-string.

+ indicates that a plus sign is to represent positive values and a minus sign is to represent negative values.

- indicates that a blank is to represent positive values and a minus sign is to represent negative values.

Mantissa

The mantissa is the decimal part of the number; it consists of 1-16 nines (each 9 representing a numeric character), and a leading, embedded, or trailing decimal point (.) or V. The decimal point indicates an actual (printed) decimal point, and the V indicates an assumed decimal point; these two characters are mutually exclusive.

E

immediately follows the mantissa and indicates that an exponent follows.

Exponent

The exponent immediately follows the second sign and consists of two consecutive 9s.

b) The PICTURE clause **must not** be specified for an internal floating-point item.

Example 2-24

External floating-point item:

```
PICTURE          -9V99E-99
PICTURE          +9999.99E+99
PICTURE          -9(16)VE+99
PICTURE          +9(16).E-99
```

7. **Alphanumeric edited data items**

Rules

a) The picture-string for an alphanumeric edited data item may contain any combination of the following symbols: A, X, 9, 0,B, / (slash).

b) In forming an alphanumeric edited picture-string, one of the following conditions must be met:

- The picture-string must contain at least one B and at least one X.
- The picture-string must contain at least one 0 and at least one X.
- The picture-string must contain at least one / (slash) and one X.
- The picture-string must contain at least one 0 and one A.
- The picture-string must contain at least one / (slash) and one A.

c) Characters from the character set of the data processing system are allowed provided that the contents are represented in standard data format.

d) Only one type of editing is performed on alphanumeric edited data items: simple insertion editing by means of the characters zero (0) and space (B) (see "Rules for Simple Insertion Editing").

8. **Numeric edited data items**

Rules

a) The picture-string for a numeric edited data item may contain any combination of the following symbols: B, / (slash), P, V, Z, 0, 9, , (comma), . (period), *, +, -, CR, DB, $.

b) The picture-string must contain at least one of the symbols 0, B, / (slash), Z, *, +, , (comma), . (period), -, CR, DB or $.

c) The maximum number of digit positions in the picture-string is 18.

d) The maximum length of a numeric edited data items is 127 characters.

e) Allowable combinations of these characters are governed by the editing rules and the permissible precedence of symbols (see "Rules for Editing..." and Table 2-7 "Precedence of Symbols used in the PICTURE Clause").

9. **Simple insertion editing**

Rules

a) In simple insertion editing, the , (comma, B (space), 0 (zero) and / (slash) are used as the insertion characters.

b) The insertion characters are counted in the size of the item, and represent the positions within the item into which they will be inserted.

Example 2-25

Category of Data	Receiving area PICTURE	Data being moved	Edited result
Numeric Edited	999,999	54321	054,321
	99B99B99	654321	65-43-21
	99B99B00	654321	43-21-00
	99/99/99	654321	65/43/21
Alphanumeric Edited	XXBXXX	123AA	12-3AA
	000X(5)	A5CD3	000A5CD3
	XX/XX	CD05	CD/05

10. **Special insertion editing**

Rules

a) Special insertion editing is performed by using the period (.) as an insertion character.

In addition to being used as the insertion character, the period is also used as the decimal point for alignment purposes.

b) The assumed decimal point (represented by the character V) and the actual (printed) decimal point cannot be used in the same picture-string.

c) Special insertion editing may be used on numeric edited data items only.

d) As a result of special insertion editing, the insertion character (decimal point) appears in the item in the same position as shown in the picture-string; thus, the insertion character is the actual decimal point. The actual decimal point is counted in the size of the item.

Example 2-26

Receiving area PICTURE	Data being moved 1)	Edited result
999.99	123&4	123.40
999.99	12&34	012.34
999.99	1&234	001.23
999.99	&1234	000.12

1) & designates the position of the assumed decimal point, which does not appear in the MOVE operation.

11. **Fixed insertion editing**

Rules

a) The editing symbols used for fixed insertion editing are: + (plus), - (minus), CR (credit), DB (debit), and $ (currency symbol).

b) Only one currency symbol and only one of the editing sign control symbols (+, -, CR, DB) may be used in a given picture-string.

c) The currency symbol may be preceded only by a plus or a minus symbol; otherwise, it must be the leftmost character.

d) The symbols CR or DB, when specified, must be either the leftmost or the rightmost character.

e) The plus or minus symbol, when specified, must be either the leftmost or the rightmost character.

f) Fixed insertion editing results in the editing character occupying the same character position in the edited item as in the picture-string.

g) The symbols CR or DB, when used, represent two character positions which are counted in the size of the data item. All fixed insertion editing characters are counted in the size of the data item.

h) Editing sign control symbols produce the results listed in Table 2-8, depending on the value of the data item.

Editing symbol in picture-string	Result	
	Data item positive or zero	Data item negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Table 2-8 Editing sign control symbols and their results

Example 2-27

Receiving area PICTURE	Data being moved 1)	Edited result
+999.99	+123&45	+123.45
+999.99	-123&45	-123.45
-999.99	+123&45	123.45
-999.99	-123&45	-123.45
$999.99CR	+123&45	$123.45
$999.99CR	-123&45	$123.45CR
$999.99DB	+123&45	$123.45
$999.99DB	-123&45	$123.45DB

1) & designates the position of the assumed decimal point, which does not appear in the MOVE operation.

12. Floating insertion editing

Rules

a) In floating insertion editing, the currency symbol ($) and the editing sign control symbols (+ and -) are used as insertion characters. These characters are mutually exclusive as floating insertion characters in the same picture-string.

Floating insertion editing is indicated in a picture-string by the use of a sequence of at least two of the allowable insertion characters to represent the leftmost numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters (, B 0 /) embedded in the sequence of floating insertion characters or to the immediate right of this sequence are part of this floating string.

b) Only two types of floating insertion editing may be specified in a picture-string.:

- Some or all of the leading numeric character positions to the left of the decimal point may be represented by insertion characters.

- All of the numeric character positions of the picture-string may be represented by insertion characters.

c) The result of floating insertion editing depends on the representation in the picture-string:

- If the insertion characters are specified only to the left of the decimal point, a single insertion character is placed into the character position which immediately precedes the decimal point, or the first non-zero digit to the left of the picture-string, and which is located inside the data represented by the insertion symbol string. The character positions preceding the insertion character are replaced with spaces.

- If each of the numeric character positions in the picture-string is represented by the insertion character, the result depends on the value of the data item concerned. If the value is zero, the entire data will contain spaces. If the value of the item is not zero, the result is the same as that occurring when the insertion characters are specified only to the left of the decimal point.

d) Every floating insertion character is counted in the size of the data item.

e) To avoid truncation of character positions, the programmer must form the picture-string for the receiving item according to the following rule:

- The minimum size of the picture-string must equal the number of nonfloating insertion characters which are used for editing in the receiving field, plus one floating insertion character.

Example 2-28

Receiving area PICTURE	Data being moved 1)	Edited Result
$$$$.99	123&12	$123.12
$$$$.99	3&12	$3.12
$$$$.99	&12	$.12
$,$$$.99	123&12	$123.12
$,$$$.99	3&12	$3.12
$,$$$.99	&12	$.12
+,+++.99	123&12	+123.12
+,+++.++	123&12	+123.12
$,$$$.99	-123&12	$123.12
-,---.99	-123&12	-123.12
$$,$$$.99	1234&56	$1,234.56
+,+++,999.99	-123456&78	-123,456.78
+,+++,+++.++	000&00	(spaces)

1) & designates the position of the assumed decimal point which does not appear in the MOVE operation.

13. **Zero suppression and replacement editing**

Rules

a) Editing by suppression of leading zeroes in numeric character positions uses the alphabetic character Z or the symbol * (asterisk) as suppression characters in the picture-string. These characters are mutually exclusive in the same picture-string. If Z is used, the replacement character is a space. If * is used, the replacement character is a space. If * is used, the replacement character is * (asterisk).

b) Zero suppression and replacement editing in a picture-string is achieved by using a string of one or more of the allowable symbols (* or Z) to represent leading numeric character positions which are to be filled with the replacement characters when the associated character positions in the data contain zeros. Any of the simple insertion characters (, B 0 /) embedded in the string of symbols or to the immediate right of the string are part of the string; each of these simple insertion characters works like an * or a Z, until a non-zero character is encountered. The simple insertion characters (, B 0 /) and fixed insertion characters ($ + -) to the left of the suppression string are not subject to the rules for zero suppression and replacement.

c) Two types of zero suppression editing may be used in a picture-string.

- Some or all of the leading numeric character positions to the left of the decimal point may be represented by suppression symbols.

- All of the numeric character positions in the picture-string may be represented by suppression symbols.

d) The result of zero suppression and replacement editing depends on the representation in the picture-string:

- If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a suppression symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point.

- If all numeric character positions in the picture-string are represented by suppression symbols and the value of the data item is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is Z, the entire data item is replaced by spaces. If the value is zero and the suppression symbol is *, all characters in the data item, except for the decimal point, are replaced by asterisks; in this case, zero suppression editing overrides the BLANK WHEN ZERO clause, if the latter is specified.

e) Each suppression character is included in the size of the data item.

Example 2-29

Receiving area PICTURE	Data being moved 1)	Editing result
ZZZZ.ZZ	0000&00	(spaces)
****.**	0000&00	****.**
ZZZZ.99	0000&00	.00
****.99	0000&00	****.00
ZZZZ.ZZ	+135&00	135.00
$**,***.**BDB	-2135&00	$*2,135.00␣DB
$BB****,**.99BBCR	-2135&00	$␣␣**21,35.00␣␣CR

1) & designates the position of the assumed decimal point which does not appear in the MOVE operation.

REDEFINES CLAUSE

Function

The REDEFINES clause allows the programmer to give an alternate description of an area of computer storage.

Format

```
level-number data-name-1 [REDEFINES data-name-2]
```

(The level number and data-name-1 are not part of the REDEFINES clause, and are shown here only for clarity.)

Rules

1. The REDEFINES clause, when used, must immediately follow data-name-1.

2. The level numbers of data-name-1 and data-name-2 must be identical but must not be 66 or 88.

3. The sizes of the items described by data-name-1 and data-name-2 may differ only in 01-level entries. In all other cases, the sizes must be equal.

4. Data-name-2 may be qualified but not indexed or subscripted.

5. The data description entry for data-name-2 must not contain an OCCURS clause; however, data-name-2 may be subordinate to a data item which contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause must not be indexed or subscripted. A data item that is subordinate to data-name-2 may contain an OCCURS clause without the DEPENDING ON phrase (see "OCCURS Clause").

6. Data-name-1 or any data item subordinate to data-name-1 may contain an OCCURS clause without the DEPENDING ON phrase. If data-name-1 contains an OCCURS clause, the size of data-name-1 is calculated by multiplying the length of one table element by the number of occurrences of the table element.

7. The data description entry of data-name-1 must immediately follow the original definition of the area of the data description entry of data-name-2. But, in the case of multiple redefinitions, other redefinitions may intervene (see Programming Consideration no. 5).

8. Data-name-1 specifies the name of the data area associated with the redefinition. Data-name-2 specifies the name of the original definition of the data area to be redefined.

9. Redefinition starts at data-name-2 and ends when a level number less than or equal to that of data-name-2 is encountered.

Programming considerations

1. The REDEFINES clause must not appear in 01-level entries in the File Section (implicit redefinition is provided by the DATA RECORDS clause in the File Description (FD) entry).

2. No entries having lower level numbers than that of data-name-1 and data-name-2 may appear between the descriptions of data-name-2 and data-name-1. Entries of level number 77 are likewise prohibited there.

3. Except for condition-name entries, the entries that redefine a storage area must not contain a VALUE clause.

4. The REDEFINES clause may be specified for an item subordinate to a redefined item, or for a data item which is subordinate to an item containing a REDEFINES clause.

5. Multiple redefinitions of the storage area are permitted but must all refer to the data-name supplied in the original definition.

6. When an area is redefined, all descriptions of that area remain in effect. For example, if A and B are two separate data items sharing the same storage area, the procedure statements MOVE ALPHA TO A or MOVE ALPHA TO B could be executed at any point in the program. In the first case, ALPHA would be moved to A and would take the form specified by the description of A. In the second case, BETA would be moved to the same physical area and would take the form specified by the description of B. If both MOVE statements were executed successively in the order specified, the value BETA would overlay the value ALPHA; however, redefinition of an area does not erase any data and does not supersede a previous description.

7. Moving a data item from A to B when B is a redefinition of A amounts to moving an item to itself, and the result of such a move is unpredictable. The same is true of the opposite type of move; that is, moving A to B when A redefines B.

8. The use of data items defined by the PICTURE and USAGE clauses within an area can be redefined. Altering the use of an area by the REDEFINES clause does not, however, change any existing data.

9. When the SYNCHRONIZED clause is specified in a data entry that is redefining a previous data entry, the user should ensure that the area being redefined begins on the proper boundary: halfword, fullword, or doubleword.

Example 2-30

```
02  ALPHA.
    03  A-1 PICTURE X(3).
    03  A-2 PICTURE X(2).
02  BETA REDEFINES ALPHA PICTURE 9(5).
02  GAMMA.
```

BETA is data-name-1; ALPHA is data-name-2. BETA redefines the area assigned to ALPHA (that is, the area occupied by A-1 and A-2). Redefinition starts at BETA and ends at the next level number 02 (the number preceding GAMMA).

Example 2-31

(Multiple Redefinition)

```
02 ALPHA PICTURE 9(3).
02 BETA REDEFINES ALPHA PICTURE X(3).
02 GAMMA REDEFINES ALPHA PICTURE A(3).
```

Example 2-32

```
01  SAMPLE-AREA-1.
    02  FIRST-DEFINITION  PICTURE 99 VALUE 12.
    02  SECOND-DEFINITION REDEFINES FIRST-DEFINITION
        USAGE COMPUTATIONAL  PICTURE S9(4).
```

In this example, FIRST-DEFINITION is a 2-byte unsigned external decimal number with the value 12. This means that the contents of the two bytes in hexadecimal is X'F1F2'. SECOND-DEFINITION is also a number, and occupies the same two bytes; but it does not have the value 12. The data in these two bytes (X'F1F2') is unchanged by the redefinition; and, since SECOND-DEFINITION is a signed, binary number, this data has the value -3598.

Example 2-33

```
01  SAMPLE-AREA-2.
    02  FIRST-DEFINITION.
        03  ALPHA                      PICTURE X(3).
        03  BETA                       PICTURE X(5).
        03  GAMMA REDEFINES BETA       PICTURE 9(5).
        03  FILLER                     PICTURE X(10).
    02  SECOND-DEFINITION REDEFINES FIRST-DEFINITION
                                       PICTURE X(18).
```

In this example, one of the items subordinate to FIRST-DEFINITION is redefined: GAMMA REDEFINES BETA. This is permitted, and is not blocked by the fact that FIRST-DEFINITION is itself later redefined by SECOND-DEFINITION.

Example 2-34

```
01  SAMPLE-AREA-3.
    02  FIRST-DEFINITION              PICTURE S9(7).
    02  SECOND-DEFINITION REDEFINES FIRST-DEFINITION.
        03  A-1 PICTURE A.
        03  N-1 REDEFINES A-1         PICTURE 9.
        03  FILLER                    PICTURE X(6).
```

In this example, one of the data items subordinate to SECOND-DEFINITION is redefined; N-1 REDEFINES A-1. This is permitted, and is not blocked by the fact that SECOND-DEFINITION itself is a redefinition.

RENAMES CLAUSE

Function

The RENAMES clause permits alternate, possibly overlapping, groupings of elementary data items. This clause assigns a new name to an item or items established by a record description. Unlike REDEFINES, the RENAMES clause does not redefine existing data descriptions but merely allows data to be accessed and/or grouped under alternative names while maintaining the previously defined data descriptions.

Format

```
66 data-name-1 RENAMES data-name-2

    [ { THRU      }          ]
    [ { THROUGH } data-name-3 ]
```

(The level number 66 and data-name-1 are not part of the RENAMES clause, and are shown only for the purpose of clarity.)

Rules

1. All entries of the RENAMES clause which refer to data items within a given data record must immediately follow the last data entry of the associated data description.

2. data-name-2 must precede data-name-3 in the record description. After each redefinition, the beginning point of the area described by data-name-3 must logically follow the beginning point of the area defined by data-name-2 (see example 2-35).

3. data-name-2 and data-name-3 must be the names of elementary items or groups of elementary data items in the associated logical record, and cannot be the same data-name.

4. The beginning of the area defined by data-name-3 must **not** lie to the left of the beginning of the area defined by data-name-2. The end of the area defined by data-name-3 **must** lie to the right of the end of the area defined by data-name-2. Hence, data-name-3 cannot be subordinate to data-name-2.

5. None of the data items within the area of data-name-2 and data-name-3, when specified, may have a variable size as described in the OCCURS clause (see "OCCURS Clause" with DEPENDING ON phrase).

6. data-name-1 cannot be used as a qualifier and can be qualified only by the names of the 01-level or FD entries.

7. data-name-2 and data-name-3 may be qualified.

8. Neither data-name-2 nor data-name-3 may contain an OCCURS clause in its respective data description, nor may it be subordinate to a data item which contains an OCCURS clause in its data description entry.
9. A 66-level entry cannot rename another 66-level entry, nor can it rename a 77-level, 88-level, or 01-level entry.

10. data-name-1 specifies an alternative definition for one or more data items.

    data-name-2 or data-name-3 specifies the data item(s) to be renamed.

11. When data-name-3 is specified, data-name-1 is a group item that includes all elementary items:

    a) starting with data-name-2 (if this is an elementary data item); or starting with the first elementary item within data-name-2 (if this is a group item).

    b) concluding with data-name-3 (if this is an elementary data item); or concluding with the last elementary item within data-name-3 (if this is a group item).

12. If data-name-3 is not specified, then data-name-2 may be either a group item or an elementary item. If data-name-2 is a group item, data-name-1 is treated as a group item; if data-name-2 is an elementary item, data-name-1 is treated as an elementary data item.

Programming consideration

More than one RENAMES clause may be written for the same data record.

Example 2-35

The following usage of the RENAMES clause is **incorrect**:

```
01  ERR-REC.
    05  GROUP-A.
        10  FIELD-1A.
            15  ITEM-1A  PICTURE XXXX.
            15  ITEM-2A  PICTURE XXX.
        10  FIELD-2A.
            15  ITEM-3A  PICTURE XXX.
            15  ITEM-4A  PICTURE XXX.
    05  GROUP-B REDEFINES GROUP-A.
        10  FIELD-1B.
            15  ITEM-1B  PICTURE XX.
            15  ITEM-2B  PICTURE XXX.
            15  ITEM-3B  PICTURE XX.
        10  FIELD-2B.
            15  ITEM-4B  PICTURE XX.
            15  ITEM-5B  PICTURE XX.
            15  ITEM-6B  PICTURE XX.
66  NEW-ERR-REC RENAMES ITEM-3A THRU ITEM-2B.
```

Although ITEM-3A precedes ITEM-2B in the record description, ITEM-2B logically precedes ITEM-3A in memory. Therefore, this example is incorrect.

Example 2-36

The following is a **correct** usage of the RENAMES clause:

```
01  RECORD.
    05  GROUP-ITEM-A.
        10  FIELD-1A.
            15  ITEM-1A  PICTURE XX.
            15  ITEM-2A  PICTURE XXX.
            15  ITEM-3A  PICTURE XX.
        10  FIELD-2A.
            15  ITEM-4A  PICTURE XX.
            15  ITEM-5A  PICTURE XX.
            15  ITEM-6A  PICTURE XX.
    05  GROUP-ITEM-B REDEFINES GROUP-ITEM-A.
        10  FIELD-1B.
            15  ITEM-1B  PICTURE XXXX.
            15  ITEM-2B  PICTURE XXX.
        10  FIELD-2B.
            15  ITEM-3B  PICTURE XXX.
            15  ITEM-4B  PICTURE XXX.
66  RECORD-NEW RENAMES ITEM-2A THRU ITEM-3B.
```

In this example, ITEM-2A precedes ITEM-3B both in the record description and logically in memory.

Example 2-37

The following example shows how a RENAMES clause may be used in an actual program:

```
01  INPUT-RECORD.
    02  ARTICLE-1.
        03  ARTICLE-NO          PIC 99.
        03  PRICE               PIC 9999.
    02  ARTICLE-2.
        03  ARTICLE-NO          PIC 99.
        03  PRICE               PIC 9999.
    02  ARTICLE-3.
        03  ARTICLE-NO          PIC 99.
        03  PRICE               PIC 9999.
    66  ART-ONE RENAMES ARTICLE-1.
    66  ART-TWO RENAMES ARTICLE-1 THRU ARTICLE-2.
    66  ART-THREE RENAMES ARTICLE-1 THRU ARTICLE-3.
```

In this case, each reference to ART-ONE would access group item ARTICLE-1; each reference to ART-TWO, the group items ARTICLE-1 and ARTICLE-2; each reference to ART-THREE, the group items ARTICLE-1, ARTICLE-2 and ARTICLE-3.

2

SIGN CLAUSE

Function

The SIGN clause specifies the position and the mode of representation of the operation sign for numeric data items.

Format

```
SIGN IS {LEADING } [SEPARATE CHARACTER]
        {TRAILING}
```

Rules

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S, or a group item containing at least one such numeric data description entry.

2. The data description entries must be described as USAGE IS DISPLAY.

3. At most one SIGN clause may apply to any given numeric data description entry.

4. If the CODE-SET clause is specified, any signed numeric data description entries must be described with the SIGN IS SEPARATE clause.

5. The SIGN clause specifies the position and the mode of representation of the operational sign. If entered for a group item, it applies to each numeric data description entry subordinate to that group. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character S; the S indicates the presence, but not the mode of representation, of the operational sign.

6. A numeric data description entry whose PICTURE contains the character S, but to which no SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character S. (For representation of the operational sign see "USAGE Clause".)

Programming considerations

1. If the SEPARATE CHARACTER phrase is omitted, then:

   a) The letter S in a picture-string is not counted in the size of the item.

   b) The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item, with the TRAILING phrase taken as the compiler's default value.

   c) For the compiler, the operational sign is C for positive and D for negative.

2. If the SEPARATE CHARACTER phrase is present, then:

   a) The letter S in a picture-string is counted in the size of the item.

   b) The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

   c) The operational signs for positive and negative are the standard data format characters + and -, respectively.p

3. Each numeric data description entry whose picture-string contains the character S is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

Example 2-38

```
.
.
.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FIELD1          PIC S999   SIGN IS LEADING SEPARATE.
01  GROUP1          USAGE IS DISPLAY.
    02  FIELD2      PIC S9(5)  SIGN IS TRAILING SEPARATE.
    02  FIELD3      PIC X(15).
    02  FIELD4      PIC S99    SIGN IS LEADING.
01  FIELD5          PIC S9(9)  SIGN IS TRAILING.

PROCEDURE DIVISION.
    MOVE ZEROES TO FIELD1,FIELD2,FIELD3,FIELD4,FIELD5.
    MOVE 3 TO FIELD4.
    MOVE -2 TO FIELD5.
    MOVE FIELD4 TO FIELD2.
    MOVE FIELD2 TO FIELD3.
    MOVE FIELD5 TO FIELD4.
```

Contents of all fields following MOVE:

(1)

decimal

+	0	0	0

FIELD1

hexadecimal

4E	F0	F0	F0

decimal

0	0	0	0	0	+

FIELD2

hexadecimal

F0	F0	F0	F0	F0	4E

decimal

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIELD3

hexadecimal

F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0

decimal

0+	0

FIELD4

hexadecimal

C0	F0

decimal

0	0	0	0	0	0	0	0	0+

FIELD5

hexadecimal

F0	F0	F0	F0	F0	F0	F0	F0	C0

(2)

decimal

+	3

FIELD4

hexadecimal

C0	F3

(3)

decimal

0	0	0	0	0	0	0	0	2-

FIELD

hexadecimal

F0	F0	F0	F0	F0	F0	F0	F0	D2

(4)

decimal

0	0	0	0	3	+

FIELD2

hexadecimal

F0	F0	F0	F0	F3	4E

(5)

decimal	0	0	0	0	3	+									
FIELD3															
hexadecimal	F0	F0	F0	F0	F3	4E	40	40	40	40	40	40	40	40	40

(6)

decimal	0-	2
FIELD4		
hexadecimal	D0	F2

2

SYNCHRONIZED CLAUSE

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on a natural boundary of the computer memory to ensure efficiency during the performance of arithmetic operations on the item.

As an extension to standard COBOL, the compiler described in this publication allows the SYNCHRONIZED clause to be specified at group level; this has the effect of aligning the elementary items subordinate to the group.

Format

```
{SYNCHRONIZED} [LEFT ]
{SYNC        } [RIGHT]
```

Rules

1. SYNC is the abbreviation of SYNCHRONIZED.

2. The LEFT or RIGHT phrase is used as comments.

3. When items are boundary-aligned because of the presence of a SYNCHRONIZED clause, it is sometimes necessary for the compiler to insert slack bytes. Slack bytes are unused character positions inserted into a record immediately ahead of an item requiring boundary alignment. Such slack bytes are included in the length of the group item containing the aligned item.

4. The actual boundary at which a synchronized item is placed depends on the phrase specified in the USAGE clause for the item.

   If the SYNCHRONIZED clause is specified, the following actions are carried out:

   For a data item with USAGE COMPUTATIONAL:

   a) For the area S9 through S9(4) in the PICTURE clause, the data item is aligned on a halfword boundary (multiple of 2).

   b) For the area S9(5) through S9(18) in the PICTURE clause, the data item is aligned on a fullword boundary (multiple of 4).

   For a data item with USAGE IS COMPUTATIONAL-1, the item is aligned on a fullword boundary.

   For a data item with USAGE IS COMPUTATIONAL-2, the item is aligned on a doubleword boundary (multiple of 8).

   For an index data item, the item is aligned on a fullword boundary.

   For a data item with USAGE DISPLAY or COMPUTATIONAL-3, the SYNCHRONIZED clause is treated as comments, as these cases do not required alignment.

5. If the SYNCHRONIZED clause is omitted for binary data items or internal floating-point items, no slack bytes are generated. However, if arithmetic operations are performed on these items, the compiler will generate the statements necessary to move these items to auxiliary fields which are properly aligned for the arithmetic operation.

6. If the SYNCHRONIZED clause is specified for a group item, then all elementary items with USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2 or INDEX will be aligned as if the SYNCHRONIZED clause had been supplied in the data descriptions of these items.

Programming considerations

1. Standard COBOL permits specification of the SYNCHRONIZED clause for elementary data items only.
   However, the compiler described in this manual allows the SYNCHRONIZED clause to be specified for group items also, with the effect described above.

2. When the SYNCHRONIZED clause is specified within a table (described by the OCCURS clause), each table element will be aligned. (This process is described under "Alignment by Insertion of Slack Bytes".)

3. When specifying the SYNCHRONIZED clause in conjunction with a REDEFINES clause, the programmer must ensure that the element being redefined is aligned (see Example 2-39).

4. The SYNCHRONIZED clause does not alter the length of an elementary data item. Each unused internal memory location (slack bytes) is included in the size of the group to which the elementary item is subordinate, and must be allowed for an internal memory allocation if the group item was the object of a REDEFINES clause (see Example 2-40).

5. All record descriptions (01-level entries) in all sections of the Data Division begin at doubleword boundaries.

6. When blocking data records that contain elementary items with the SYNCHRONIZED clause specified, the user must add the necessary slack bytes to ensure proper alignment after the first data record within the block. (This process is described under "Alignment by Insertion of Slack Bytes").

7. For the purpose of aligning elementary items with USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, specified in the LINKAGE section, all 01-level elementary items are assumed to be aligned on doubleword boundaries. Consequently, the user must ensure that these operands are appropriately aligned in the USING phrase when he writes a CALL statement.

Example 2-39

The following example requires that A be aligned on a fullword boundary:

```
02  A  PICTURE X(4).
02  B  REDEFINES A PICTURE S9(9) USAGE COMP SYNC.
```

Example 2-40

```
01  RECORD.
    02  A.
        03  G PICTURE X(5).
        03  H PICTURE S9(9) SYNC USAGE COMP.
    02  B   REDEFINES A.
        03  I PICTURE X(12).
```

Here, elementary item G occupies 5 bytes, and elementary item H occupies 4 bytes. The SYNCHRONIZED and USAGE clauses indicate that the elementary data item H is aligned on fullword boundary; elementary item H is therefore preceded by 3 slack bytes. As data item A as a whole occupies 12 bytes, the subject of the REDEFINES clause (data item B) must also occupy 12 bytes.

USAGE CLAUSE

Function

The USAGE clause specifies the data format in which a data item is stored in the computer memory.

Format

```
USAGE IS  [DISPLAY                ]
          [COMPUTATIONAL          ]
          [COMP                   ]
          [COMPUTATIONAL-1        ]
          [COMP-1                 ]
          {COMPUTATIONAL-2        }
          [                       ]
          [COMP-2                 ]
          [COMPUTATIONAL-3        ]
          [COMP-3                 ]
          [INDEX                  ]
```

Rules

1. COMP is short for COMPUTATIONAL.
   COMP-1 is short for COMPUTATIONAL-1.
   COMP-2 is short for COMPUTATIONAL-2.
   COMP-3 is short for COMPUTATIONAL-3.

2. When the USAGE clause is omitted for an elementary item, or for a group item, its data format is assumed to be DISPLAY.

3. For the description of the various categories of data see chapter 1.

Programming considerations

1. The USAGE clause may be specified at any data description level. If the USAGE clause is specified at group level, it is applicable to all of the elementary data items of that group.

2. An elementary item which was described with USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2 or COMPUTATIONAL-3 represents a value which is used in arithmetic operations, and must therefore be numeric. If any of these phrases is specified for a group item, such a phrase will refer only to the elementary items in that group; the group item as such must not be used in arithmetic operations.

3. The USAGE of an elementary item must not contradict the USAGE of the group item to which the elementary item belongs.

4. The USAGE clause does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clauses of the operands referred to.

5. The internal representation of the numeric data items is shown in Table 2-9.

The **rules for the INDEX phrase** are discussed in chapter 6, "Table Handling".

6. **DISPLAY phrase**

Rules

a) The use of the DISPLAY phrase applies to the following types of elementary data items: alphabetic, alphanumeric-edited, numeric-edited, external-decimal, and external floating-point.

b) The type of elementary item for which the DISPLAY phrase is written is defined by the picture-string in the respective PICTURE clause.

c) External decimal data items are described under Programming Consideration a), external floating-point data items under Programming Consideration b). In addition, all data items are also described under "PICTURE Clause".

d) The DISPLAY phrase specifies that the data item is to be stored in standard data format; that is, in character form, with one character per 8-bit byte. Each character position of the data item is represented by one byte, as specified in the appropriate picture-string of the PICTURE clause.

Programming considerations

a) External decimal data items are internally represented as follows:

Each digit of a number is represented by a single byte. The four high-order bits of each byte are the zone portion. The zone portion of the low-order or high-order byte (depending on the SIGN clause) represents the sign of the number, assuming that a sign exists. The four low-order bits contain the value of the digit.

The maximum length of an external decimal item is 18 digits.

b) External floating-point items consist of a mantissa, which represents the decimal part of the number, and an exponent with the base 10.

The value of an external floating-point item is calculated by multiplying the mantissa by 10 to the power of exponent. The magnitude of a number represented by a floating-point field must be greater than 5.4×10^{-79} and must not exceed 72×10^{75}

An external floating-point item, when used as a numeric operand, is checked at object time and is converted into an internal floating-point item. It is used in this form in arithmetic operations (see notes on COMPUTATIONAL-1 and COMPUTATIONAL-2).

Example 2-41

DISPLAY phrase

The USAGE for all examples is DISPLAY.

Data category	Value	PICTURE description	Internal representation 1)
alphabetic	ABCD	AAAA.	C1 C2 C3 C4
alphanumeric	A1B2	XXXX.	C1 F1 C2 F2
alphanumeric edited	123AB	XXBXXX.	F1 F2 40 F3 C1 C2
numeric edited	54321	99.999	F5 F4 6B F3 F2 F1
numeric			
external decimal	+1234	9999	F1 F2 F3 F4
	+6879	S9999	F6 F8 F7 C9
	-6879	S9999	F6 F8 F7 D9
external floating point	6879	+99.99E-99	4E F6 F8 6B F7 F9 C5 40 F0 F2
	.6879	+99.99E-99	4E F6 F8 6B F7 F9 C5 60 F0 F2

1) Each box represents one byte.

7. COMPUTATIONAL phrase

Rules

a) This phrase applies to binary data items.

b) The PICTURE clause of a binary data item must contain no other characters but 9's, the operational sign S, the implied decimal point V, and one or more P's (see "PICTURE Clause").

c) The COMPUTATIONAL phrase indicates that the value of a data item is stored in binary format.

d) The data items are stored in a halfword, fullword, or doubleword, and are aligned only if the SYNCHRONIZED clause was specified.

e) The number of positions in the picture-string supplied in the PICTURE clause must be equal to the number of digits in the numeric value.

Programming considerations

a) The storage requirements for a binary item vary, depending on the number of decimal digits specified in its PICTURE clause, as follows:

Decimal digits in the PICTURE clause	Bytes required in memory	Alignment
1-4	2	halfword
5-9	4	fullword
10-18	8	fullword

b) The leftmost bit of a binary data item in internal memory is the operational sign. The remaining bits represent the value.

For examples of the COMPUTATIONAL phrase see Table 2-9, "Internal representation of numeric data items".

8. **COMPUTATIONAL-1 phrase**

Rules

a) This phrase applies to internal floating-point items, which are equivalent to external floating-point items in terms of capacity and application (see "Data categories").

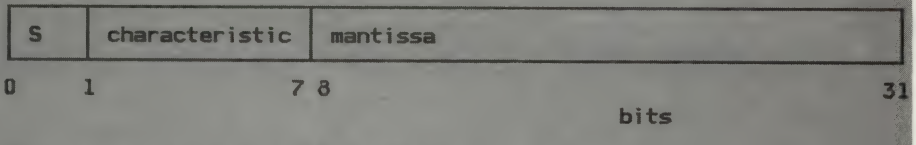
b) For a COMPUTATIONAL-1 data item, the PICTURE clause is prohibited.

c) The COMPUTATIONAL-1 phrase indicates that a data item is stored in short precision floating-point format.

d) A COMPUTATIONAL-1 data item has a length of 4 bytes and is aligned on a fullword boundary if the SYNCHRONIZED clause is supplied.

Programming considerations

a) A COMPUTATIONAL-1 data item is represented in memory as follows:



Here, S is the sign of the mantissa.
Characteristic = exponent + 64

b) An internal short precision floating-point item permits representation with an accuracy of seven decimal digits.

c) The following applies to the value that may be represented in a COMPUTATIONAL-1 data item:

value = 0 or the absolute value of it may range from $5.4 * 10^{-79}$ to $7.2 * 10^{75}$

For examples of the COMPUTATIONAL-1 phrase, see Table 2-9, "Internal Representation of Numeric Data Items".

9. COMPUTATIONAL-2 phrase

Rules

a) This phrase applies to internal floating-point items, which are equivalent to external floating-point items in terms of capacity and application (see "Data categories").

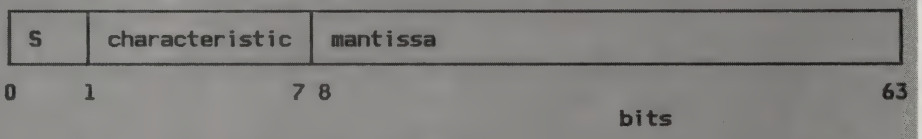
b) For a COMPUTATIONAL-2 item, the PICTURE clause is prohibited.

c) The COMPUTATIONAL-2 phrase indicates that a data item is to be stored in long-precision floating-point format.

d) A COMPUTATIONAL-2 item has a length of 8 bytes and is aligned on a doubleword boundary if the SYNCHRONIZED clause is specified.

Programming considerations

a) A COMPUTATIONAL-2 data item is represented in memory as follows:



Here, S is the sign of the mantissa.
Characteristic = exponent + 64

b) A long-precision internal floating-point item permits representation with an accuracy of 16 decimal digits.

c) The following applies to the value that may be represented in a COMPUTATIONAL-2 data item:

value = 0 or the absolute value of it may range from $5.4 * 10^{-79}$ to $7.2 * 10^{75}$

For examples of the COMPUTATIONAL-2 phrase, see Table 2-9, "Internal representation of numeric data items".

10. **COMPUTATIONAL-3 phrase**

Rules

a) The COMPUTATIONAL-3 phrase indicates that the data item is stored in internal decimal format (i.e. in packed form).

b) The PICTURE clause of a COMPUTATIONAL-3 item may contain no characters other than 9s, the operation sign S, the assumed decimal point V, and one or more Ps (see "PICTURE Clause").

Programming considerations

a) Internal decimal data items are represented by 2 digits per byte; the sign is contained in the four low-order bits of the rightmost byte.

The sign position for internal decimal items whose PICTURE clause contains no S is occupied by a bit configuration which may be interpreted as positive; this bit configuration does not, however, represent an overpunch (see "COB1 User's Guide" [7]).

b) An internal decimal item may contain any of the digits from 0 to 9 and a sign. Its value must not be greater than 18 digits.

For examples of the COMPUTATIONAL-3 phrase, see Table 2-9, "Internal representation of numeric data items".

Form	PICTURE	USAGE and SIGN entry	Value in external representation	Value in internal representation 4)	Bytes required	Conversion for arithmetic operation	Boundary alignment if synchronized
External decimal (zoned)	9999	DISPLAY	1234	F1 F2 F3 F4	digit	Conform to format of other operands or COMPUTATIONAL-3.	
	9999		1234	F1 F2 F3 F4			
	S9999		+1234	F1 F2 F3 C4 1) 2)			
	S9999		−1234	F1 F2 F3 D4 1) 2)			
	S9999	DISPLAY SIGN TRAILING	1234+	F1 F2 F3 C4			
			1234−	F1 F2 F3 D4			
	S9999	DISPLAY SIGN TRAILING SEPARATE	1234+	F1 F2 F3 F4 4E	+ 1 byte for sign		
			1234−	F1 F2 F3 F4 60			
	S9999	DISPLAY SIGN LEADING	+1234	C1 F2 F3 F4			
			−1234	D1 F2 F3 F4			
	S9999	DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4	+ 1 byte for sign		
			−1234	60 F1 F2 F3 F4			
Internal decimal (packed)	9999	COMP-3	+1234	01 23 4F 2)	2 digits per byte, except for low-order byte which contains a digit and a sign	No. Except when other operand is binary and conversion to binary would be more advantageous.	None
	9999		−1234	01 23 4F 2)			
	S9999		+1234	01 23 4C 2)			
	S9999		−1234	01 23 4D 2)			

Form	PICTURE	USAGE and SIGN entry	Value in external representation	Value in internal representation 4)	Bytes required	Conversion required for arithmetic operation	Boundary alignment if synchronized
Binary	S9999	COMP	+1234	04 D2	2 bytes for 1-4 digits 4 bytes for 5-9 digits 8 bytes for 10-18 digits	No. Except when used in mixed-form computations to maintain common formats, or if COMPUTATIONAL-3 would be more advantageous.	
	S9999		-1234	FB 2E	2		Halfword 3)
Ext. float. point	+99.99E-99	DISPLAY	+12.34E+2	4E F1 F2 6B F3 F4 C5 40 F0 F2	1 byte per character	Yes. To external floating point	None
Int. floating point	None allowed	COMP-1	+12.34E+2	43 4D 20 00	4	No	Word
	None allowed	COMP-2	-12.34E-2	C0 1F 97 24 74 53 8E F3	8	No	Doubleword

Table 2-9 Internal representation of numeric data items

1) One byte per digit and one byte for the sign (see EBCDI Code).

2) Mode of sign representation:
   F - non-printable plus sign (treated as an absolute value)
   C - internal equivalent of plus sign
   D - internal equivalent of minus sign.

3) See rules for binary data items.

4) Each box represents one byte.

VALUE CLAUSE

Function

The VALUE clause defines the initial value of a data item in the WORKING-STORAGE SECTION, the value of a printable data item of the REPORT SECTION, or the value or range of values associated with a condition-name.

Format 1 of the VALUE clause is specified to define the initial value of a data item in the WORKING-STORAGE SECTION or the value of a printable data item in the REPORT SECTION.

Format 2 of the VALUE clause is specified to define the value or range of values associated with a condition-name.

Format 1

```
[VALUE IS literal]
```

Rules for format 1

1. The literals specified in the formats of the VALUE clause may be replaced by figurative constants.

2. All numeric literals in the VALUE clause of a data item must have a length which is within the number of positions specified by the PICTURE clause, and must not have a value which would require truncation of non-zero digits.

3. Non-numeric literals in the VALUE clause of a data item must not exceed the size specified in the PICTURE clause.

4. A signed numeric literal must be associated with a PICTURE clause which provides a signed numeric picture-string.

5. In the WORKING-STORAGE SECTION, the VALUE clause may be used to specify the initial value of any data item; in this case, the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is omitted from the description of an item, the initial value of that item is unpredictable.

6. In the REPORT section, if a report group elementary item description entry with a VALUE clause contains no GROUP INDICATE clause, the printable data item will assume the specified value whenever its report group is printed; however, if the GROUP INDICATE clause is also supplied, then the printable data item assumes the specified value only if certain conditions become true during the execution of the program.

Programming considerations for format 1

1. The VALUE clause is prohibited for external floating-point data items.

2. The VALUE clause must not be specified for any item whose size, whether explicitly or implicitly, is variable.

3. The VALUE clause must not conflict with other clauses in the data description of an item or in the data description within the hierarchy of an item. The following rules are applicable:

   If the category of the item being described is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a WORKING-STORAGE item, the literal is standard-aligned into the data item.

   If the category if the item being described is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric.

4. A data item is set to its initial value irrespective of whether a BLANK WHEN ZERO clause or JUSTIFIED clause was specified.

5. The VALUE clause must not be specified in sections other than the WORKING-STORAGE or REPORT section.

6. The VALUE clause must not be stated in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an entry containing an OCCURS clause.

7. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause or in any entry that is subordinate to an entry containing a REDEFINES clause.

8. Within a given record description, the VALUE clause must not be stated in a data description entry that is preceded by a data description entry in which an OCCURS clause with a DEPENDING ON phrase appears.

9. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal; in this case, the group area is initialized regardless of the individual elementary or group items contained within the group. The VALUE clause cannot be supplied at the subordinate levels within the group.

10. The VALUE clause must not be written at the group level for a group containing items described with the JUSTIFIED or SYNCHRONIZED clauses, or with a USAGE clause other than USAGE IS DISPLAY.

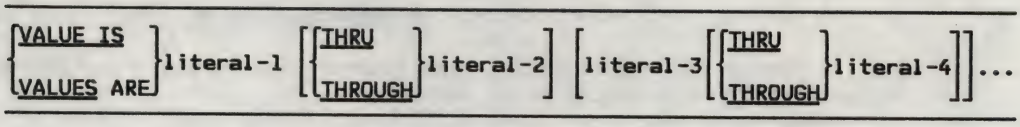
Example 2-42

(for format 1)

```
77  FIELD PICTURE IS AA VALUE IS "AA".
77  FILLER PICTURE IS XX VALUE SPACES.
```

Here the value of FIELD is initialized to AA, and the unnamed data item (FILLER) is initialized to spaces.

Format 2



Rules for format 2

1. The VALUE clause in format 2 may only be used in conjunction with condition-names (level-number 88).

2. The specified literals may be replaced by figurative constants.

3. All numeric literals must have a length which is within the number of positions specified by the PICTURE clause, and must not have a value which would require truncation of non-zero digits.

4. Nonnumeric literals must not exceed the size specified in the PICTURE clause.

5. A signed numeric literal must be associated with a PICTURE clause which provides a signed numeric picture-string.

6. The THRU/THROUGH phrase is only allowed for a condition-name which is associated with a numeric conditional variable (see rule 7).

   When the THRU/THROUGH phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

7. Level-number 88 is assigned to entries which define condition-names associated with a conditional variable; these are called condition-name entries. A conditional variable is a data item that is followed by one or more condition-name entries. A condition-name is a name that is assigned a value or range of values when the program is executed. Therefore, a condition-name may be either "true" or "false" at object time. A condition-name is not a data item, and it requires no memory space (see under "Condition-Name Conditions", Description of the Use of Condition-Names).

8. The THRU/THROUGH phrase assigns a range of values to the specified condition-name.

Programming consideration for format 2

1. The VALUE clause must not be specified for external floating-point data items.

2. The VALUE clause must not be specified for items whose size, whether explicitly or implicitly, is variable.

3. The VALUE clause must not conflict with other clauses in the data description of an item or in the data description within the hierarchy of an item. The following rules are applicable:

   If the category of the item being described is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a WORKING-STORAGE item, the literal is standard-aligned into the data item.

   If the category of the item being described is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric.

4. The initial value of a data item is not affected by the specification of a BLANK WHEN ZERO or JUSTIFIED clause.

5. The VALUE clause is only allowed in the FILE, WORKING-STORAGE and LINKAGE sections. It must not be specified in the REPORT SECTION.

6. The VALUE clause and the condition-name itself are the only two entries permitted in a condition-name entry.

7. The condition-name entries for a particular conditional variable must follow that conditional variable. That is, an 88-level entry must be preceded either by an entry for the conditional variable or another 88-level entry.

8. The literal category in a condition-name entry must be consistent with the data category of the conditional variable. For example, when the PICTURE clause for a conditional variable contains AAA, then each condition-name assigned to that conditional variable must have 1 to 3 alphabetic characters, consisting entirely of letters and/or spaces.

9. Every condition-name may be qualified by the name of its conditional variable and/or by the qualifier of the conditional variable.

10. A condition-name may be subordinate to a data item requiring indexing or subscripting. In this case, the condition-name, once it is written in the Procedure Division, must be indexed or subscripted according to the requirements of the associated conditional variable.

11. A condition-name must not be associated with the following data items:

    a) a data item of level number 66

    b) a group item whose data items are described with JUSTIFIED, SYNCHRONIZED, or USAGE (except USAGE IS DISPLAY)

    c) an index data item (see chapter 6, "Table Handling").

Example 2-43

Format 2

```
02  CITIES PICTURE 9.
    88  BERLIN VALUE 1.
    88  HAMBURG VALUE 2.
    88  MUNICH VALUE 3.
    88  COLOGNE VALUE 4.
```

Here, CITIES is the conditional variable, and BERLIN, HAMBURG, MUNICH, and COLOGNE are the condition-names. If a statement IF MUNICH GO TO TEST-C were written in the Procedure Division, then the value of the conditional variable CITIES would be compared to the value 3; this statement would be equivalent to the statement IF CITIES IS EQUAL TO 3 GO TO TEST-C.

Example 2-44

Format 2

```
02  AGE PICTURE 99.
    88  TWENTIES VALUE 20 THRU 29.
    88  THIRTIES VALUE 30 THRU 39.
```

If the statement IF TWENTIES... were to be written in the Procedure Division, the value of the conditional variable AGE would be compared to the values 20, 21, ... and 29. This statement would be equivalent to the statement IF AGE NOT LESS THAN 20 AND NOT GREATER THAN 29...

Example 2-45

Format 2

```
02  NAME-OF-DAY PICTURE X(3).
    88  BEGINNING-WEEK VALUE "MON" "TUE" "WED".
    88  END-OF-WEEK    VALUE "THRU" "FRI".
    88  WEEKEND        VALUE "SAT" "SUN".
```

If the statement IF BEGINNING-OF-WEEK... were to be written in the Procedure Division, the conditional variable NAME-OF-DAY would be compared with "MON", "TUE" and "WED". This statement would be equivalent to IF NAME-OF-DAY IS EQUAL TO "MON" OR "TUE" OR "WED".

2.5 PROCEDURE DIVISION

2.5.1 General Description

The Procedure Division contains the specific instructions for solving a given data processing problem. COBOL instructions are written in the form of statements.

A **statement** is a syntactically valid combination of words and symbols, beginning with a COBOL verb, for example:

```
MOVE A TO B.
```

Several statements may be combined into a sentence; groups of sentences, into paragraphs; and one or more paragraph, into a section.

Normally, a COBOL statement refers to programmer-defined data or procedures by using data-names or procedure-names. References to user-defined words must be unique (see under "Qualification").

A logical subset of the program, consisting of one or more successive paragraphs or one or more successive sections of the Procedure Division, is called a "procedure". A procedure-name is a word which is used for referring to a paragraph or a section; it consists of a paragraph-name (which may be qualified by a section-name) or a section-name.

Up to 8000 procedure names are allowed within a source program, including those that are generated by the compiler for SORT, Report Writer and ENTER statements.

Two types of procedures are provided within the Procedure Division:

a) Declarative procedures, which cannot be executed within the normal sequence of statements in the Procedure Division.

b) Non-declarative procedures that contain statements for normal execution when there are no special exceptional conditions.

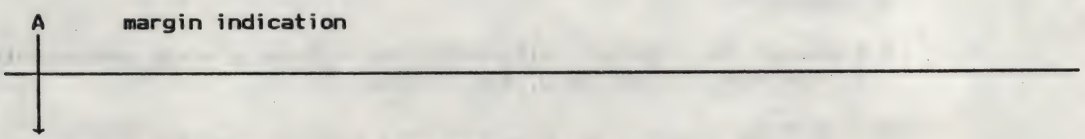
The execution of the compiled object program begins with the first Procedure Division statement which follows the declaratives. Statements are executed in the order in which they are present at compile time, unless some other order is specified in the rules for a given statement.

If program segmentation is used, the programmer must divide the entire Procedure Division into named sections. Program segmentation is discussed in chapter 9, "Segmentation".

2.5.2 Structure

GENERAL FORMAT

Format 1

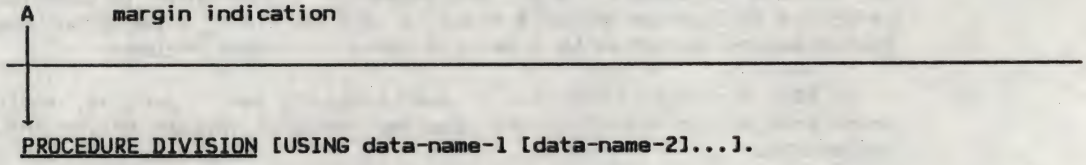



```
PROCEDURE DIVISION [USING data-name-1 [data-name-2]...].

[ DECLARATIVES.
{section-name SECTION.USE statement.[paragraph-name.[sentence]...]...}...
  END DECLARATIVES.]

[{section-name SECTION [segment number].[paragraph-name.[sentence]...]...}...]
```

Format 2




```
PROCEDURE DIVISION [USING data-name-1 [data-name-2]...].

[{paragraph-name. [sentence]...}...]
```

General rules

1. The Procedure Division must begin with the header PROCEDURE DIVISION, followed by a period and a space, unless Program Communication is being used. In this case, the Procedure Division header in a called program may optionally include the USING phrase preceding the period (see chapter 7, "Program Communication").

2. The Procedure Division header is followed, optionally, by the declarative portion containing declarative procedures, which is followed, in its turn, by nondeclarative procedures. Each of these procedures consists of statements, sentences, paragraphs, and/or sections in a syntactically valid format. The end of the Procedure Division (and the physical end of the source program) is the point in a COBOL source program which is not followed by any further procedures and statements.

3. For a description of the declarative subdivision, see "Declarative Subdivision of the Procedure Division".

4. If sections are used within the Procedure Division, format 1 must be applied. Otherwise, format 2 may be used.

5. A **section** consists of a section header followed either by zero, one, or more successive paragraphs. (The section header consists of a section-name, followed by the word SECTION and a period; if program segmentation is desired, a space and a segment-number followed by a period may be inserted after the word SECTION.) A section can end in one of three ways: immediately before the next section-name; at the end of the Procedure Division; or in the declaratives subdivision of the Procedure Division immediately before the next section or at the keywords END DECLARATIVES.

   Multiply-defined section-names, or multiply-defined paragraph-names within a section, will not be treated as errors by the compiler so long as they are not referenced.

   Paragraph- and section-names are restricted to a length of 30 bytes.

6. A **paragraph** consists at a minimum of a paragraph-name followed by a period and a blank. It may be followed by one or more sentences. A paragraph ends immediately before the next paragraph or section, or at the end of the Procedure Division, or in the declarative subdivision of the Procedure Division immediately before the next paragraph or section or at the keywords END DECLARATIVES.

7. If one paragraph is in a section, then all of the paragraphs must be in sections.

8. A **sentence** consists of one or more statements, optionally separated by semicolons, and is terminated by a period, followed by spaces.

9. A **statement** consists of a syntactically valid combination of words and symbols, and must begin with a COBOL word.

2.5.3 Arithmetic Expressions

Function

Arithmetic expressions allow the user to combine arithmetic operations.

Format

An **arithmetic expression** may be one of the following:

a) An identifier of a numeric elementary item.

b) A numeric literal.

c) Numeric identifiers and numeric literals separated by arithmetic operators.

d) Two arithmetic expressions separated by an arithmetic operator or an arithmetic expression enclosed in parentheses.

Any arithmetic expression may be preceded by a unary plus (+) or a unary minus (-).

2.5.3.1 Arithmetic Operators

The following **operators** may be used in arithmetic expressions:

Binary arithmetic operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary arithmetic operator	Meaning
+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1

An arithmetic operator must always be preceded by a space and followed by a space. A unary operator must always be preceded by a space or a left parenthesis; it must be followed by a space when it precedes a data-name and must not be followed by a space when it precedes a numeric literal.

2.5.3.2 Rules for the Formation and Evaluation of Expressions

1. An arithmetic expression may begin only with a left parenthesis, a unary +, a unary -, an identifier, or a literal; and it may end only with a right parenthesis or a variable (identifier or literal).

2. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression.

3. Table 2-10 shows the valid combination of operators, variables, and parentheses in arithmetic expressions.

First Symbol	Second Symbol				
	identifier, literal	arithmetic operator	unary operator	(	)
identifier, literal	-	P	-	-	P
arithmetic operator	P	-	P	P	-
unary + or -	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

Table 2-10 Valid symbol combinations in arithmetic expressions

Legend

P indicates that the two symbols may appear consecutively.

- indicates that the two symbols may not appear consecutively.

4. Parentheses may be used in arithmetic expressions in order to indicate the order in which the elements are to be evaluated.

5. Expressions within parentheses are evaluated first. When nested parentheses are used, evaluation proceeds from the innermost to the outermost set of parentheses.

6. When no parentheses are used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   a) Unary plus or minus (evaluated first)
   b) Exponentiation
   c) Multiplication and division
   d) Addition and subtraction (evaluated last)

7. If consecutive operations have the same hierarchical level, they are evaluated from left to right.

Programming considerations

1. Parentheses may be used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to alter the normal order of execution.

2. Arithmetic expressions are used in arithmetic and conditional statements.

Example 2-46

Expression: A + (B - C) * D

Evaluation:
1. B - C (denote result by x)
2. x * D (denote result by y)
3. A + y (final result)

Example 2-47

Expression: A + ((B/C) + (D * * E) * F) - G

Evaluation:
1. B/C (denote result by z)
2. D * * E (denote result by x)
3. x * F (denote result by y)
4. z + y (denote result by a)
5. A + a (denote result by b)
6. b - G (final result)

2.5.4 Conditions

General Description

A condition enables the object program to select one of two alternate paths of execution, depending upon the truth value of a test. A distinction is made between simple and compound conditions.

Simple Conditions:

a) Relation condition
b) Class condition
c) Condition-name condition
d) Switch-status condition
e) Sign condition

Each of these conditions may be negated by the preceding word "NOT".

Compound Conditions:

Compound condition is the name given to simple conditions which are placed within sets of parentheses and connected by the logical operators AND or OR.

2.5.4.1 Simple Conditions

CONDITION-NAME CONDITION

Function

The condition-name condition causes a conditional variable to be tested to determine whether or not its value is equal to one of the values associated with a specified condition-name. (See VALUE clause for additional information.)

Format

```
condition-name
```

Rules

1. condition-name specifies the condition-name to be used in the test.

2. If a condition-name is associated with a single value, then the related test is true, only if the value corresponding to the condition-name equals the value of its associated conditional variable.

3. If the condition-name is associated with one or more ranges of values, then the conditional variable is tested to determine whether or not its value falls in the range, including the end values.

4. The condition-name condition is a shorthand form of the relation condition (see example).

   See also "SET Statement", Format 4.

Example 2-48

```
02  PAY-CLASS PICTURE 9.
    88 HOURLY   VALUE 1.
    88 WEEKLY   VALUE 2.
    88 MONTHLY  VALUE 3.
       ...
    IF HOURLY GO TO HOUR-PROCEDURE.
```

Here, PAY-CLASS is a conditional variable, and HOURLY, WEEKLY, and MONTHLY are condition-names. If the current value of PAY-CLASS is 1, the result of the test in the IF statement is true. Otherwise, the result is false.

As noted above, the condition-name is a shorthand form of the relation condition. The following statement, which contains a relation condition, is equivalent to the above IF statement:

```
IF PAY-CLASS = 1 GO TO HOUR-PROCEDURE.
```

CLASS CONDITION

Function

The class condition determines whether an operand is numeric, alphabetic, alphabetic in uppercase or alphabetic in lowercase, or whether it consists entirely of characters from a character set declared with class-name in the CLASS clause of the SPECIAL-NAMES paragraph.

Format

```
identifier IS [NOT] {NUMERIC          }
                    {ALPHABETIC       }
                    {ALPHABETIC-LOWER }
                    {ALPHABETIC-UPPER }
                    {class-name       }
```

Rules

1. The identifier must be a data item described implicitly or explicitly with USAGE IS DISPLAY or COMPUTATIONAL-3.

2. The identifier specifies the data item to be tested.

3. NUMERIC, ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER and class-name (possibly negated by NOT) indicate which characteristic is to be tested.

4. Identifier is treated as numeric when its contents consist of a combination of the digits 0 to 9 (with or without sign).

5. Identifier is treated as alphabetic when its contents consist of any combination of the characters A to Z and/or a to z and the space character.

6. Identifier is treated as alphabetic in lowercase (ALPHABETIC-LOWER) when its contents consist of a combination of the lowercase letters a-z and the space character.

7. Identifier is treated as alphabetic in uppercase (ALPHABETIC-UPPER) when its contents consist of a combination of the uppercase letters A to Z and the space character.

8. Identifier corresponds to class-name when its contents consist solely of a combination of those characters which were defined by means of class-name in the SPECIAL-NAMES paragraph.

9. Identifier cannot be tested as numeric if it has been defined in the data description entry either as an alphabetic data item or as a group item consisting of signed data elements ("S" symbol).

   If the picture-string of identifier is unsigned (PIC 9), identifier will not be considered numeric unless its contents are numeric and the operational sign is F.

If the picture-string contains an operational sign definition (PIC S9), identifier will not be considered numeric unless its contents are numeric and a valid operational sign is present (represented by C, D, or F).

10. An identifier cannot be tested as alphabetic, alphabetic in lowercase, or alphabetic in uppercase, or compared to class-name, if it has been defined as numeric in the data description entry.

11. Table 2-11 lists all permissible formats of the class condition.

Type of identifier	Tests allowed	
Alphabetic or group item with signed data element	ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, class-name	NOT ALPHABETIC, NOT ALPHABETIC-LOWER, NOT ALPHABETIC-UPPER, NOT class-name
Alphanumeric or alphanumeric-edited or numeric-edited	ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, NUMERIC, class-name	NOT ALPHABETIC, NOT ALPHABETIC-LOWER, NOT ALPHABETIC-UPPER, NOT NUMERIC, NOT class-name
Numeric	NUMERIC	NOT NUMERIC

Table 2-11 Valid forms of the class text

CLASS CONDITION

Function

The class condition determines whether an operand is numeric, alphabetic, alphabetic in uppercase or alphabetic in lowercase, or whether it consists entirely of characters from a character set declared with class-name in the CLASS clause of the SPECIAL-NAMES paragraph.

Format

```
identifier IS [NOT] {NUMERIC          }
                    {ALPHABETIC       }
                    {ALPHABETIC-LOWER }
                    {ALPHABETIC-UPPER }
                    {class-name       }
```

Rules

1. The identifier must be a data item described implicitly or explicitly with USAGE IS DISPLAY or COMPUTATIONAL-3.

2. The identifier specifies the data item to be tested.

3. NUMERIC, ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER and class-name (possibly negated by NOT) indicate which characteristic is to be tested.

4. Identifier is treated as numeric when its contents consist of a combination of the digits 0 to 9 (with or without sign).

5. Identifier is treated as alphabetic when its contents consist of any combination of the characters A to Z and/or a to z and the space character.

6. Identifier is treated as alphabetic in lowercase (ALPHABETIC-LOWER) when its contents consist of a combination of the lowercase letters a-z and the space character.

7. Identifier is treated as alphabetic in uppercase (ALPHABETIC-UPPER) when its contents consist of a combination of the uppercase letters A to Z and the space character.

8. Identifier corresponds to class-name when its contents consist solely of a combination of those characters which were defined by means of class-name in the SPECIAL-NAMES paragraph.

9. Identifier cannot be tested as numeric if it has been defined in the data description entry either as an alphabetic data item or as a group item consisting of signed data elements ("S" symbol).

   If the picture-string of identifier is unsigned (PIC 9), identifier will not be considered numeric unless its contents are numeric and the operational sign is F.

If the picture-string contains an operational sign definition (PIC S9), identifier will not be considered numeric unless its contents are numeric and a valid operational sign is present (represented by C, D, or F).

10. An identifier cannot be tested as alphabetic, alphabetic in lowercase, or alphabetic in uppercase, or compared to class-name, if it has been defined as numeric in the data description entry.

11. Table 2-11 lists all permissible formats of the class condition.

Type of identifier	Tests allowed	
Alphabetic or group item with signed data element	ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, class-name	NOT ALPHABETIC, NOT ALPHABETIC-LOWER, NOT ALPHABETIC-UPPER, NOT class-name
Alphanumeric or alphanumeric-edited or numeric-edited	ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, NUMERIC, class-name	NOT ALPHABETIC, NOT ALPHABETIC-LOWER, NOT ALPHABETIC-UPPER, NOT NUMERIC, NOT class-name
Numeric	NUMERIC	NOT NUMERIC

Table 2-11 Valid forms of the class text

SWITCH-STATUS CONDITION

Function

The switch-status condition tests the setting of a user or task switch. The implementor-name specified and its associated ON or OFF value must appear in the SPECIAL-NAMES paragraph of the Environment Division.

Format

condition-name

Rule

The result of the test is true if the switch is set to the position corresponding to the condition-name.

The status of a switch can be changed by using a SET statement, format 3 (see "SET Statement, Format 3").

RELATION CONDITION

Function

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, or an arithmetic expression.

Format

```
{identifier-1           }                          {identifier-2           }
{literal-1              }                          {literal-2              }
{arithmetic-expression-1} relational-operator      {arithmetic-expression-2}
{index-1                }                          {index-2                }
```

Rules

1. The first operand of a relational condition is called the subject of the condition, and the second operand is called the object of the condition. The operands must be written according to the following rules:

   a) The subject and object may not both be literals.

   b) The subject and object must have the same data format, except when two numeric operands are compared.

2. Relational-operator must be one of the operators listed in Table 2-12. It must be preceded and followed by one space.

Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] > 1)	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] < 1)	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to
IS GREATER THAN OR EQUAL TO IS ≥	Self-explanatory
IS LESS THAN OR EQUAL TO IS ≤	Self-explanatory

Table 2-12 Relational operators

1) These special characters are not underlined to avoid being mistaken for other characters, e.g. ≥ (greater or equal to).

3. The relational operator specifies the type of comparison to be made in a relation test.

4. The following rules describe comparisons between numeric operands, comparisons between nonnumeric operands, and comparisons between index-names and/or index data items. In compare operations, a group item is treated as a nonnumeric data item.

5. **Comparison of numeric operands**

   When two numeric operands are compared, their algebraic values are compared; their lengths (that is, the number of digits they contain) are not significant.

   Unsigned numeric operands are considered to be positive for purposes of comparison.

   Zero is considered to be a unique value, regardless of sign.

   Comparison of two numeric operands is permitted, regardless of the data formats in their respective USAGE clauses.

   Example 2-49

```
-50          IS LESS THAN    +5
+75       IS GREATER THAN +5
-100         IS LESS THAN    -10
-0           IS EQUAL TO     +0
```

6. **Comparison of nonnumeric operands**

   When two nonnumeric operands are compared, or when a numeric operand is compared with a nonnumeric operand, the comparison is made with respect to the binary collating sequence of the PROGRAM COLLATING (see "OBJECT-COMPUTER Paragraph").

   If one of the operands is numeric, it must be an integral data item or integral literal. Furthermore:

   a) If the nonnumeric operand is an **elementary item** or a nonnumeric literal, the numeric operand will be treated as if it had been transferred to an alphanumeric elementary item equal in size to the numeric item, and as if the contents of this alphanumeric elementary item had been compared with the nonnumeric operand (see "MOVE Statement" and "PICTURE Clause, Symbol P").

   b) If the nonnumeric operand is a **group item**, the numeric operand will be treated as if it had been transferred to a group item equal in size to the numeric data item, and as if the contents of this group item had then been compared with the nonnumeric operand (see "MOVE Statement" and "PICTURE Clause, Symbol P").

   c) A numeric operand which is not an integer cannot be compared with a nonnumeric operand.

   Another important factor in a nonnumeric comparison is the size of the operands. The size of an operand is equal to the total number of characters within it. There are two cases to consider: the comparison of operands of equal size, and the comparison of operands of unequal size. In either case, each operand may be up to 131071 bytes long.

   a) Comparison of operands of equal size

If the operands are of equal size, the comparison takes place as follows: The object program compares characters in corresponding character positions; it starts at the high-order (that is, leftmost) end, and continues until it encounters two unequal characters or until it reaches the end of the operands.

If the object program encounters a pair of unequal characters, it determines which character has a higher position in the collating sequence. The operand containing the higher character is considered to be the greater operand.

If all characters compare equally through to the last pair, the operands are considered equal.

Example 2-50

In the following examples the binary collating sequence of the EBCDIC character set is assumed, i.e. either PROGRAM COLLATING SEQUENCE has been omitted or PROGRAM COLLATING SEQUENCE IS NATIVE has been specified.

Relationship	Reason
"123" IS GREATER THAN "ABC"	1 (the 1st character in the 1st operand) is greater than A (the 1st character in the 2nd operand).
"SMYTH" IS GREATER THAN "SMITH"	Y (the 3rd character in the 1st operand) is greater than I (the 3rd character in the 2nd operand).
"ABC" IS EQUAL to "ABC"	All characters compare equally.

b) Comparison of operands of unequal size

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

Example 2-51

Relationship	Reason
"CAR" IS GREATER THEN "AUTO"	C (the 1st character of the 1st operand) is greater than A (the 1st character of the 2nd operand).
"SMITH" is less than "SMITHY"	SMITH is padded with blanks as follows: SMITH␣. The blank (sixth character in first operand) is less than Y (sixth character in second operand).

c) Comparisons involving index-names and/or index data items.

Table 6-2 in chapter 6, "Table Handling", gives a detailed list of the allowable comparisons involving index-names and/or index data items, and indicates the items compared in each case.

7. All allowable comparisons are shown in Table 2-13.

First operand	Second operand													
	GR	AL	AN	ANE	NE	FC2) NNL	ZR NL	ED	BI	ID	EF	IF	IN	IDI
Group element (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN		
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Alphanumeric edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Numeric edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Fig. constant (FC)2) and nonnumeric literal (NNL)	NN	NN	NN	NN	NN			NN			NN			
Fig. constant ZERO (ZR) and numeric literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU	NU	IN3)	
External decimal (ED)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	IN3)	
Binary (BI)	NN						NU	NU	NU	NU	NU	NU	IN3)	
Internal decimal (ID)	NN						NU	NU	NU	NU	NU	NU	IN3)	
Ext. floating point (EF)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU		
Int. floating point (IF)	NN						NU	NU	NU	NU	NU	NU		
Index name (IN)							IN3)	IN3)	IN3)	IN3)			TI	ID
Index data item (IDI)													ID	ID

Table 2-13 Valid comparisons between various operands 1)

1) Function values within table

NN = comparison as described for nonnumeric operands
NU = comparison as described for numeric operands
TI = comparison as described for two index-names (see "Table Handling")
IN = comparison as described for index-name and numeric integer (see "Table Handling")
ID = comparison as described for index data item and index-name or other index data items (see "Table Handling").

2) FC includes all figurative constants except ZERO.

3) Valid only if the numeric item is an integer.

SIGN CONDITION

Function

The sign condition determines whether or not the algebraic value of a numeric operand (that is, an item described as numeric) is less than, greater than, or equal to zero.

Format

```
{identifier            } IS [NOT] {POSITIVE}
{arithmetic-expression }          {NEGATIVE}
                                  {ZERO    }
```

Rules

1. identifier or arithmetic-expression identifies the operand to be tested.

2. POSITIVE, NEGATIVE, or ZERO specifies the test to be made.

3. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

2.5.4.2 Compound Conditions

Function

A compound condition consists of a combination of two or more simple conditions.

Format

```
condition {AND} [NOT] condition [{AND} [NOT] condition] ...
          {OR}                  [{OR}                ]
```

Rules

1. condition specifies a simple condition.
2. Parentheses may be used within a compound condition, in order to eliminate ambiguities or to alter the normal order of execution.
3. The simple conditions within a compound condition are separated from each other by logical operators, according to specified rules. The logical operators must be preceded by a space and followed by a space.
4. A compound condition may comprise up to 60 simple conditions.
5. Table 2-14 lists the logical operators and their meanings.

Operator	Meaning	Example
OR	Logical inclusive or (either or both)	The expression A OR B is true if A is true, or B is true, or both A and B are true.
AND	Logical conjunction (both)	The expression A AND B is true only if both A and B are true.
NOT	Logical negation	The expression "NOT" A is true only if A is false.

Table 2-14 Logical operators

6. The ways in which conditions and logical operators may be combined are shown in Table 2-15.

First symbol	Second symbol					
	Simple condition	OR	AND	NOT	(	)
Simple condition	-	P	P	-	-	P
OR	P	-	-	P	P	-
AND	P	-	-	P	P	-
NOT	P	-	-	-	P	-
(	P	-	-	P	P	-
)	-	P	P	-	-	P

Table 2-15 Valid symbol pairs of conditions and logical operators 1)

1) P indicates that the two symbols may appear consecutively.
- indicates that the two symbols may not appear consecutively.

7. **Evaluation rules**

The evaluation of compound conditions starts with the innermost set of parentheses and continues through to the outermost set of parentheses.

If the order of evaluation is not determined by parentheses, then the expression is evaluated in the following hierarchical order:

Arithmetic expressions

Relational operators

NOT conditions

AND and its surrounding conditions are evaluated from left to right.

OR and its surrounding conditions are evaluated last, also proceeding from left to right.

If consecutive expressions have the same hierarchical level, they are evaluated from left to right.

Example 2-52

Consider this expression:

A IS NOT GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE

This expression is evaluated as if the following parentheses had been supplied:

(A IS NOT GREATER THAN B) OR (((A+B) IS EQUAL TO C) AND (D IS POSITIVE)).

Example 2-53

Table 2-16 shows some of the relationships between logical operators and simple conditions.

Operands	Value of A 1)	True	False	True	False
	Value of B 1)	True	True	False	False
Combinations	NOT A	False	True	False	True
	A AND B	True	False	False	False
	A OR B	True	True	True	False
	NOT (A AND B)	False	True	True	True
	NOT A AND B	False	True	False	False
	NOT (A OR B)	False	False	False	True
	NOT A OR B	True	True	False	True

Table 2-16 Use of logical operators

1) A and B represent simple conditions.

2.5.4.3 Implied Subjects and Relational Operators

Function

When a compound condition is written without parentheses, any relation condition except the first may be abbreviated as follows:

- the subject of the relation condition may be omitted.
- the subject and relational operator of the relation condition may be omitted.

However, the compiler permits the use of parentheses, namely in relation subjects and relation objects which are arithmetic expressions, and in order to influence the sequence in which the logical operators AND and OR are analyzed.

Format

Format of implied subject

```
...subject relational-operator object { AND } [NOT] relational-operator object...
                                      { OR  }
```

Format of Implied Subject and Relational Operator

```
...subject relational-operator object { AND } [NOT] object...
                                      { OR  }
```

Rules

1. Within a sequence of relation conditions, both forms of abbreviation may be used. The effect of using such abbreviations is the same as if the omitted subject were replaced by the most recently stated subject, or the omitted relational-operator were replaced by the most recently stated relational-operator.

2. Ambiguity may result from using NOT in conjunction with abbreviations. In this case, NOT will be interpreted as part of a relational operator if followed immediately by <, LESS, =, EQUAL, > or GREATER. In all other cases it will be considered to be a logical operator. A NOT appearing in front of a left parentheses will remain in effect up to the associated right parenthesis (see example 2-58).

Example 2-54

Implied subjects

```
IF X = Y OR > W OR < Z
```

is equivalent to

```
IF X = Y OR > W OR X < Z
```

In this example, the implied subject is the most recently stated subject, i.e. X.

Example 2-55

Implied subjects and relational operators

```
IF X = Y OR Z OR W
```

is equivalent to

```
IF X = Y OR X = Z OR X = W
```

In this example, the implied subject is the most recently stated subject, i.e. X; and the implied relational operator is the most recently stated operator, i.e. =.

Example 2-56

Implied subject, and implied subject with relational operator

```
X = Y AND > Z OR A
```

is equivalent to

```
X = Y AND X > Z OR X > A
```

Here, since X is the only stated subject, it is substituted in both simple conditions. The most recently stated operator, >, is substituted in the third simple condition.

Example 2-57

```
A > B AND NOT > C OR D
```

is equivalent to

```
A > B AND NOT A > C OR NOT A > D
```

or

```
((A > B) AND (A NOT > C)) OR (A NOT > D)
```

Example 2-58

```
A NOT = "A" AND NOT ("B" OR NOT "C")
```

is equivalent to

```
A NOT = "A" AND NOT (A NOT = "B" OR NOT A NOT = "C")
```

or

```
A NOT = "A" AND A = "B" AND A NOT = "C"
```

or

```
A = "B".
```

2.5.5 Arithmetic Statements

Rules

1. All identifiers used in arithmetic statements must be defined as numeric data in the Data Division.

2. All literals used in arithmetic statements must be numeric. They may be floating-point literals.

3. The maximum size of any operand (identifier or literal) is 18 decimal digits.

4. The maximum size of all results after decimal-point alignment is 18 decimal digits.

5. When several operands occurring in an arithmetic statement are "overlapped" in a hypothetical data item, aligned on their decimal points, then the maximum size of the data item required (i.e. the composite of operands) is 18 decimal digits (see "ADD Statement" and "SUBTRACT Statement").

6. A maximum of 30 operands may be supplied in one arithmetic statement.

7. The format of any data item involved in computations (for example, an addend, a subtrahend, or a multiplier) cannot contain editing symbols. Operational signs and implied decimal points are not considered to be editing symbols.

8. Identifiers which are used only to receive the result of an arithmetic statement (for example, the identifier used with the GIVING phrase) may be numeric-edited items (see "GIVING Phrase").

9. Condition-names cannot appear as operands.

Programming considerations

1. The operands need not have the same data description; any necessary conversion and decimal point alignment is supplied throughout the computation (see "MOVE Statement", Rules for numeric moves).

2. If the sending or receiving fields of an arithmetic statement, or of an INSPECT, MOVE, SET, STRING or UNSTRING statement, share the same storage area (that is, if the operands overlap), execution of that statement produces unpredictable results.

3. The results are also unpredictable if the identifiers contain any data other than numeric data at object time.

   Note

   The programmer determines whether the object program should treat spaces (X'40') in external decimal items as valid numeric data, by specifying a special operand as a compile-time parameter (see "COB1 User's Guide" [7]).

   If the input operands for an arithmetic statement do not contain valid numeric data, a data error will probably occur at object time.

4. In evaluating arithmetic statements, the compiler generates a number of arithmetic operations. Depending on the relationship between the various operands, the compiler will generate one or more intermediate result fields. These intermediate result fields are retained until required for solving the final result of that statement.

   Table 2-17 shows the number of the integer and decimal digits which are stored in the result field according to the operation executed. From this table, it is possible to determine the optimum operand size for the desired precision to be achieved in the statement. On the basis of the decimal places contained in each operand, and by reference to the formulae supplied in the table, the programmer may determine the exact number of positions which the compiler will make available to the result field. However, the result placed in the result field in decimal-point aligned. Hence decimal-point alignment is likewise important in determining the precision of the result.

   Other considerations affecting the results of arithmetic operations are (on this, see also Table 2-17):

   a) If the ROUNDED phrase is specified, then the value of Fd (decimal places in the result) will be Fd+1.

   b) In all additions or subtractions where an operand has a USAGE declared as COMPUTATIONAL or COMPUTATIONAL-3, "i" (the calculated number of integer places) is increased by 1.

Statement Type	Operation	Decimal Places in Intermediate Result (d)	Integer Places in Intermediate Result (i)	IF i+d > 30 digits Decimals	Integers
Arithmetic	ADD or SUBTRACT (+) or (-)	greater of Ad or Bd	greater of (Ai+1) or (Bi+1)	Fd	30-Fd
	MULTIPLY (*)	Ad plus Bd	Ai plus Bi	Fd	30-Fd
	DIVIDE (/)	greater of (Fd+1) or Bd	Bi plus Ad	Bd-Ad	Bi+Ad
	EXPONENTIATION (**)	Fd	(total digits in final result field) less Fd	(not appli-cable)	(not appli-cable)
IF or PERFORM	ADD or SUBTRACT (+) or (-)	greater of Ad or Bd	30-d	(not cable)	(not appli-cable)
	MULTIPLY (*)	Ad+Bd	30-d		
	DIVIDE (/)	Bd	30-d		
	EXPONENTIATION (**)	12	18		

Table 2-17 Calculating the integer and decimal places in intermediate results

i = calculated integer places

Ai = integers in first operand 1)

Bi = integers in second operand 2)

Fi = integers in final result

d = calculated decimal places

Ad = decimal places in first operand 1)

Bd = decimal places in second operand 2)

Fd = decimal places in final result

1) Divisor in the case of division

2) Dividend in the case of division

2.5.6 Options in Arithmetic Statements

The arithmetic statements are used to perform computations. The four basic arithmetic operations are specified by the ADD, SUBTRACT, MULTIPLY and DIVIDE statements. These operations may be combined in the COMPUTE statement.

The phrases covered below are common to all types of arithmetic statements.

CORRESPONDING PHRASE

The CORRESPONDING phrase enables the user to write one statement to perform computations on several elementary items of the same name in different groups.

1. The word CORRESPONDING may be abbreviated as CORR.
2. All identifiers must refer to group items.
3. The descriptions of the identifiers must not contain data items with level numbers 66, 77, or 88, or the USAGE IS INDEX clause.
4. Pairs of data items correspond if the following conditions exist; all other items are ignored for the operation:
5. Both data items have the same name and qualification, up to, but not necessarily including, identifier-1 and identifier-2.
6. None of the data items is declared with FILLER.
7. In case of MOVE CORRESPONDING, at least one of the data items is an elementary item; both data items are elementary items in case of ADD CORRESPONDING or SUBTRACT CORRESPONDING.
8. A data item that is subordinate to identifier-1 or identifier-2 and that is defined with a REDEFINES, OCCURS, or USAGE IS INDEX clause will be ignored ; any items which are subordinate to such items are also ignored. However, identifier-1 or identifier-2 may be defined with REDEFINES or OCCURS clauses or be subordinate to data items defined with REDEFINES or OCCURS clauses.

Example 2-59

In this example, elementary items in EMPLOYEE-RECORD are subtracted from corresponding items in PAYROLL-CHECK.

Procedure Division statement:

```
SUBTRACT CORRESPONDING EMPLOYEE-RECORD FROM PAYROLL-CHECK.
```

Data Division entries:

```
01  EMPLOYEE-RECORD.
    02  EMPLOYEE-NUMBER.
        03  PLANT-LOCATION...
        03  CLOCK-NUMBER.
            04  SHIFT-CODE...
            04  CONTROL-NUMBER...
    02  INCOME.
        03  HOURS-WORKED...
        03  PAY-RATE...
    02  FICA-RATE...
    02  DEDUCTIONS...
```

```
01  PAYROLL-CHECK.
    02  EMPLOYEE-NUMBER.
        03  CLOCK-NUMBER...
        03  FILLER...
    02  DEDUCTIONS.
        03  FICA-RATE...
        03  WITHHOLDING-TAX...
        03  PERSONAL-LOANS...
    02  INCOME.
        03  HOURS-WORKED...
        03  PAY-RATE...
    02  NET-PAY...
    02  EMPLOYEE-NAME.
        03  SHIFT-CODE...
```

According to the rules for the CORRESPONDING phrase, the following subtractions take place:

1st operand	2nd operand
HOURS-WORKED OF INCOME OF EMPLOYEE-RECORD	HOURS-WORKED OF INCOME OF PAYROLL-CHECK
PAY-RATE OF INCOME OF EMPLOYEE-RECORD	PAY-RATE OF INCOME OF PAYROLL-CHECK

The following items are not subtracted, for the reasons stated:

Item	Reason
EMPLOYEE-NUMBER	Item not elementary item in either group
PLANT-LOCATION OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Name does not appear under PAYROLL-CHECK
CLOCK-NUMBER OF EMPLOYEE-NUMBER	Item is not elementary in one group
SHIFT-CODE OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK
CONTROL-NUMBER OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Name does not appear under PAYROLL-CHECK
INCOME	Name is not elementary in either group
TAX-RATE OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK
DEDUCTIONS	Item not elementary in one group

GIVING PHRASE

Rules

1. The word GIVING is followed by an identifier which may be a numeric-edited item since it is not itself involved in the calculation.

2. When the GIVING phrase is supplied, the result of the arithmetic operation is assigned to the identifier specified.

3. The result stored in the identifier replaces the previous contents of the identifier. Therefore, it is not necessary to reset the identifier to zero.

Example 2-60

```
ADD A B GIVING C.
```

The value C is set to the sum of A + B, and A and B are not changed.

ROUNDED PHRASE

Rules

1. If, after decimal-point alignment, the number of places following the decimal point in the result of an arithmetic operation is greater than the number of decimal places provided in the resultant identifier, truncation is performed according to the size of this identifier. If rounding is specified, the absolute value of the last significant digit of the resultant identifier is incremented by 1 if the most significant digit of those to be truncated is greater than or equal to 5.

2. If rounding is not desired, but truncation is necessary, the last digit of the resultant-identifier remains unaltered.

3. If the least significant digits of a resultant-identifier are represented by P in the picture-string for this identifier, rounding or truncation takes place on the basis of the rightmost digit for which internal storage is allocated (see example).

ROUNDED is assumed for COMPUTATIONAL-1 or COMPUTATIONAL-2 result fields, and need not be specified for them.

Example 2-61

Calculated result 1)	Description of result field	Description after rounding	Result without rounding
03&2627	PIC 99 PIC 99.9 PIC 99.99 PIC 99.999	03 03.3 03.26 03.263	03 03.2 03.26 03.262
123788&6	PIC S999PPP	124000	123000

1) & represents the operational decimal point.

SIZE ERROR PHRASE

A size error condition exists if, after decimal-point alignment, the integer digits in the computed result exceed the number of places provided for them and thus cause an overflow.

Rules

1. The SIZE ERROR entry contains an imperative statement which specifies what actions are to be taken in the event of a size error.

2. The size error condition applies only to the final results of an arithmetic operation and not to intermediate results, except in the case of the MULTIPLY and DIVIDE statements.

3. If the ROUNDED phrase is specified, rounding takes place before the size error check.

4. If the SIZE ERROR phrase is specified and overflow occurs, the imperative statement in the SIZE ERROR entry is executed. The value of the resultant-identifier is not altered.

5. If the SIZE ERROR phrase is not specified and overflow occurs, the value of the resultant-identifiers is unpredictable.

6. Whether or not the SIZE ERROR phrase is specified, the values of the resultant-identifiers for which no overflow occurs remain unaffected by any size errors occurring for other resultant-identifiers during execution of this operation (see "ADD Statement" and "SUBTRACT Statement").

7. If an ADD statement with the CORRESPONDING phrase or a SUBTRACT statement with the CORRESPONDING phrase causes overflow in one of the steps of the operation, the imperative statement specified in the SIZE ERROR phrase is not executed until all of the additions or subtractions are completed.

8. Division by zero always causes a size error condition.

   For COMPUTATIONAL-1 or COMPUTATIONAL-2 data items, division by zero will cause the imperative statement in the SIZE ERROR phrase to be executed.

9. In the case of division with ON SIZE ERROR, it is still possible for a DIVIDE-ERROR to occur since no test is made for quotient overflow (only for division by zero).

Example 2-62

Description of result item: PIC 99

Statement:

```
ADD A TO B ON SIZE ERROR GO TO PROC-A.
```

```
Current Value of A:   44
Current Value of B:   72
Calculated Result:   116 (size error condition triggered)
Result of Execution: The GO TO statement is executed.
```

Since the SIZE ERROR phrase is specified, the imperative statement GO TO PROC-A is executed. The result item is unchanged.

2.5.7 Overlapping Operands

The following rule applies for all statements:

If a sending and a receiving field occupy the same memory location, or a part thereof, but are not defined in the same data description entry, the results of statement execution will be unpredictable. In the case of certain statements, the results of statement execution will be unpredictable even if the sending and receiving fields are defined in the same data description entry.

Further information on this point can be found under the individual statements.

2.5.8 Incompatible Data

Apart from the class condition test, each time a data item is addressed in the Procedure Division the following applies:

If the contents of a data item are incompatible with the data class defined in its PICTURE clause, the results of the operation will be unpredictable.

Programming consideration

Every operation involving a numeric data item which may possibly have nonnumeric contents (e.g. due to a redefinition of the data item or following a MOVE statement using a group item as operand) should be preceded by the IF NUMERIC class test. The operation can only be performed successfully if the class test yields the truth value TRUE.

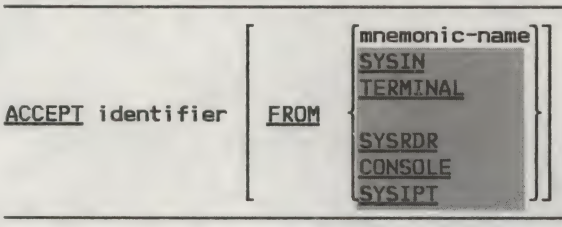
2.5.9 Statements

ACCEPT STATEMENT

Function

The ACCEPT statement transfers small quantities of data to a data item. The data are either read from a system file or are made available by the compiler or operating system.

Format 1



Rules for format 1

1. The identifier can be a group item or an alphabetic, alphanumeric, external decimal or external floating-point data item.

2. The mnemonic-name must be specified in the SPECIAL-NAMES paragraph and is associated with one of the following implementor-names:

   SYSIPT,
   SYSIN, SYSRDR, TERMINAL,
   CONSOLE;
   job-variable-name (BS2000 job variable),
   COMPILER-INFO,
   CPU-TIME, PROCESS-INFO, TERMINAL-INFO.

3. Data is moved left-justified to the area indicated by identifier, regardless of the picture-string associated with the identifier. Incoming data is not edited, and no error checking is performed.

4. SYSIPT, SYSIN, SYSRDR, TERMINAL, CONSOLE or mnemonic-name specifies the system file from which data is to be read.

   SYSIPT refers to the system file of that name.

   SYSIN, SYSRDR, and TERMINAL refer to the system file SYSDTA (normally assigned to the data terminal).

   CONSOLE refers to the system console.

5. When entered for a job-variable-name, mnemonic-name addresses the associated operating system job variable which is to be read in.

6. When entered for COMPILER-INFO, CPU-TIME, PROCESS-INFO, or TERMINAL-INFO, mnemonic-name specifies the information which is to be requested.

   COMPILER-INFO refers to information provided by the compiler.

   CPU-TIME, PROCESS-INFO, and TERMINAL-INFO refer to information provided by the operating system.

7. If the FROM phrase is omitted, data is read from the system file SYSIPT.

8. The execution of ACCEPT statements and the structure of the information provided for the individual functions are described in the BS2000 COB1 User's Guide.

Programming considerations for format 1

1. If the system file specified for an ACCEPT statement is the same as one designated for a READ statement, the results will be unpredictable.

2. Any ACCEPT statement for job-variable will be rejected at object time with an error message if the job variables do not exist in the operating system.

Example 2-63 (format 1)

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
    COMPILER-INFO IS COMPINFO
    PROCESS-INFO IS PROCINFO
    TERMINAL-INFO IS TERMINFO
    JV-COBJV IS JOBVARIABLE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  COMPILER-INFORMATION.
    02  COMPILER-VERSION                PIC X(10).
    02  DATE-OF-COMPILATION.
        03  JJ                          PIC XX.
        03  FILLER                      PIC X.
        03  MM                          PIC XX.
        03  FILLER                      PIC X.
        03  TT                          PIC XX.
    02  TIME-OF-COMPILATION             PIC X(8).
    02  PROGRAM-NAME                    PIC X(30).
01  TASK-INFORMATION.
    02  TASK-TYPE                       PIC X.
        88  BATCH-TASK                          VALUE "B".
        88  DIALOG-TASK                         VALUE "D".
    02  TASK-SEQUENCE-NUMBER            PIC 9(4).
    02  USER-IDENTIFICATION             PIC X(8).
    02  ACCOUNT-NUMBER                  PIC X(8).
    02  PRIVILEGE-IDENTIFIER            PIC X.
        88  SYSTEM-ADMINISTRATOR                VALUE "S".
        88  USER                                VALUE "U".
    02  OPERATING-SYSTEM-VERSION        PIC X(10).
01  TERMINAL-INFORMATION.
    02  TERMINAL-NAME                   PIC X(8).
    02  NUM-CHARS-LINE                  PIC 9(5).
    02  NUM-LINES-SCREEN                PIC 9(5).
    02  NUM-CHARS-SCREEN                PIC 9(5).
    02  DEVICE-TYPE                     PIC X(4).
01  DATA-INPUT                          PIC X(256).
PROCEDURE DIVISION.
    ACCEPT COMPILER-INFORMATION FROM COMPINFO.
    ACCEPT TASK-INFORMATION FROM PROCINFO.
    ACCEPT TERMINAL-INFORMATION FROM TERMINFO.
    ACCEPT DATA-INPUT FROM JOBVARIABLE. ----------------------------(01)
```

(01) = The contents of the job variable with link-name *COBJV are moved to the field DATA-INPUT.

Example 2-64 (format 1)

```
ACCEPT INPUT1 FROM TERMINAL.
```

This ACCEPT statement requests data from the system file SYSDTA and moves them to the data item indicated by INPUT1.

Format 2

```
ACCEPT identifier  FROM { DATE        }
                        { DAY         }
                        { DAY-OF-WEEK }
                        { TIME        }
```

Rules for format 2

1. The identifier must not be an alphabetic elementary item or a floating-point data item (COMP-1 or COMP-2).

2. The ACCEPT statement causes the requested information to be moved to the data item specified by the identifier, in accordance with the rules governing the MOVE statement. DATE, DAY, DAY-OF-WEEK and TIME are special data items and thus are not described in the source program.

Programming considerations for format 2

1. DATE consists of the elementary items "year" of century, "month" of year, and "day" of month. The order of these implied elementary items is as follows, from left to right: year, month, day. Thus, for example, April 1, 1977 would be shown as 770401. DATE, if referenced in a COBOL program, is analyzed as if it had been declared as an unsigned, elementary numeric integer data item, six digits in length (PIC 9(6)).

2. DAY consists of the elementary items "year" of century, and "day" of year. The order of these implied elementary items is as follows, from left to right: year, day. Thus, for example, April 1, 1977 would be shown as 77091. DAY, if referenced in a COBOL program, is analyzed as if it had been declared as an unsigned, elementary numeric integer data item, five digits in length (PIC9(5)).

3. DAY-OF-WEEK consists of a single elementary item containing the day of the week. If referenced in a COBOL program, DAY-OF-WEEK is analyzed as if it had been declared as an unsigned, elementary numeric integer data item with a length of one digit.
   With DAY-OF-WEEK, the value 1 is Monday, 2 is Tuesday, ... , 7 is Sunday.

4. TIME consists of the elementary items "hours", "minutes", "seconds" and "hundredths of seconds". TIME is based on the 24-hour clock; thus 2:41 pm, for example, is rendered as 14410000. If referenced in a COBOL program, TIME is analyzed as if it had been declared as an unsigned elementary numeric integer data item, 8 digits in length (PIC 9(8)). The minimum value for TIME is 00000000, the maximum value is 23595900. The last two digits are not supplied by the system, and are therefore always set to zero.

ADD STATEMENT

Function

The ADD statement adds two or more numeric operands and stores the result.

Format 1 of the ADD statement stores the sum in one of the operand fields. More than one addition may be expressed by specifying more than one result field in the same ADD statement.

Format 2 of the ADD statement stores the sum in a separate result field.

Format 3 of the ADD statement uses the CORRESPONDING phrase.

Format 1

```
ADD {identifier-1} [identifier-2]
    {literal-1   } [literal-2   ] ... TO identifier-m [ROUNDED]

    [identifier-n [ROUNDED]]...

    [ON SIZE ERROR imperative-statement]
```

Rules for format 1

1. Each identifier must refer to an elementary numeric item.

2. The composite of operands determined by using all of the operands in a given statement must not be more than 18 digits (see "Arithmetic Statements").

3. The values of the operands preceding the word TO are added together; and the sum is added to the current value of identifier-m, identifier-n, ... The result is stored in identifier-m, identifier-n, ...

For further rules, see "Options in Arithmetic Statements"; the ROUNDED phrase and the SIZE ERROR phrase are described in this section.

Example 2-65 (format 1)

Statement	Result field PICTURE IS	Calculation
ADD A, B TO C, D		A + B + C stored in C A + B + D stored in D
ADD A, B, C TO D	S9999V99	A + B + C + D stored in D as nnnnVnn
ADD A, 14 TO C ROUNDED	99999	A + 14 + C stored in C as nnnnn; rounded if necessary

Format 2

```
ADD {identifier-1} [{identifier-2}]... TO [identifier-3]
    {literal-1   }  {literal-2   }        [literal-3   ]

    GIVING identifier-m [ROUNDED] [identifier-n [ROUNDED]]...

    [ON SIZE ERROR imperative-statement]
```

Rules for format 2

1. Each identifier preceding the GIVING phrase must refer to an elementary numeric item.

2. identifier-m may refer either to an elementary numeric item or to an elementary numeric-edited data item.

3. The composite of operands determined by using all of the operands in a given statement, except for the data items which follow the word GIVING, must not be more than 18 digits (see "Arithmetic Statements").

4. When the GIVING phrase is used, the values of the operands preceding the word GIVING are added together; then, the sum is stored as the new value of identifier-m, identifier-n...

For further rules, see "Options in Arithmetic Statements"; the GIVING, ROUNDED and SIZE ERROR phrases are described in this section.

Example 2-66 (format 2)

Statement	Result field PICTURE IS	Calculation
ADD A, B, C GIVING D.	S9999.99	A + B + C stored in D as SnnnnVnn
ADD A, B, 43.6 GIVING D ON SIZE ERROR GO TO O-FLOW.	99V99	A + B + 43.6 stored in D. If the integer result is greater than 2 digits, the size error condition occurs; and the GO TO statement specified in the SIZE ERROR phrase is executed.

Format 3

```
ADD {CORR          } identifier-1 TO identifier-2 [ROUNDED]
    {CORRESPONDING}

    [ON SIZE ERROR imperative-statement]
```

Rules for format 3

1. Each identifier must refer to a group item.
2. The composite of operands which is determined separately for each pair of corresponding data items must not be greater than 18 digits (see "Arithmetic Statements").
3. When the CORRESPONDING phrase is used, elementary data items within the the first operand (identifier-1) are added to the corresponding elementary data items in the second operand (identifier-2 ...). The results are stored in the items of the second operand.

For further rules, see "Options in Arithmetic Statements"; the CORRESPONDING, ROUNDED, and SIZE ERROR phrases are described in this section.

Example 2-67 (format 3)

Refer to the description of the CORRESPONDING phrase for an example of the use of this option.

ALTER STATEMENT 1)

Function

The ALTER statement modifies one or more GO TO statements, thereby altering a previously defined sequence of operations.

Format

```
ALTER  procedure-name-1 TO [PROCEED TO] procedure-name-2

      [procedure-name-3 TO [PROCEED TO] procedure-name-4]...
```

Rules

1. procedure-name-1, procedure-name-3 etc. must be names of paragraphs which contain only one sentence consisting of a GO TO statement without DEPENDING phrase.

2. procedure-name-2, procedure-name-4 etc. must be paragraph names or section names in the Procedure Division.

3. During the execution of the object program, the ALTER statement modifies the GO TO statement specified under procedure-name-1, procedure-name-3 ..., replacing the entry points of the GO TO statement by procedure-name-2, procedure-name-4 ..., respectively (see "GO TO Statement").

Programming considerations

1. A GO TO statement in a section whose segment number is greater than or equal to 50 must not be referenced by an ALTER statement in a section with a different segment number.

2. A GO TO statement in a section whose segment number is less than 50 may be referenced by an ALTER statement in any section, even if the GO TO statement thus referenced is in a program segment which has not yet been called for execution.

1) Bearing in mind the future development of COBOL, the ALTER statement should be avoided when writing new programs.

Example 2-68

```
          .
          .
          .
X-SWITCH.
    GO TO X1.
          .
          .
          .
X1.
    ALTER X-SWITCH TO PROCEED TO X2.
          .
          .
          .
    GO TO X.
          .
          .
          .
X2.
```

The statements in this example are executed as follows:
When the X-SWITCH paragraph is executed for the first time, it passes control to the paragraph names X1. The ALTER statement changes the GO TO statement into GO TO X2. Thus, the next time paragraph X-SWITCH is executed, control is passed to the paragraph named X2.

Example 2-69

```
ALTER FINISH TO PROCEED TO INITIAL.
          .
          .
          .
FINISH. GO TO.
```

In this case, the ALTER statement changes the GO TO statement into GO TO INITIAL. Thus, when the GO TO statement is executed, it will pass control the the procedure named INITIAL.

2

COMPUTE STATEMENT

Function

The COMPUTE statement is used to assign the value of a data item, literal, or arithmetic expression to a data item.

Format

```
COMPUTE identifier-1 [ROUNDED] [identifier-2 [ROUNDED]]... = {identifier-3
                                                             {literal-1
                                                             {arithmetic-expression}

     [ON SIZE ERROR imperative-statement]
```

Rules

1. identifier-1, identifier-2... must refer to an elementary numeric item or an elementary numeric-edited data item.

2. identifier-3 must refer to an elementary numeric item.

3. The "arithmetic-expression" in the COMPUTE statement permits the use of any meaningful combination of identifiers (which must satisfy the general rules for data-names in simple arithmetic operations), literals, and arithmetic operands; if necessary, they may also be in parentheses (see also "Arithmetic Expressions").

4. If identifier-3 or literal-1 is specified, the value of identifier-1 is set equal to the value of identifier-3 or literal-1.

5. When an arithmetic-expression is used, the value of that arithmetic-expression is first calculated and then stored as the new value of identifier-1, identifier-2.

6. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite of operands and receiving items which are imposed by the ADD and SUBTRACT statements.

7. Up to 50 data-names may be specified in a COMPUTE statement.

For further rules, see under "Options in Arithmetic Statements"; the ROUNDED and SIZE ERROR phrases are described in this section.

Example 2-70

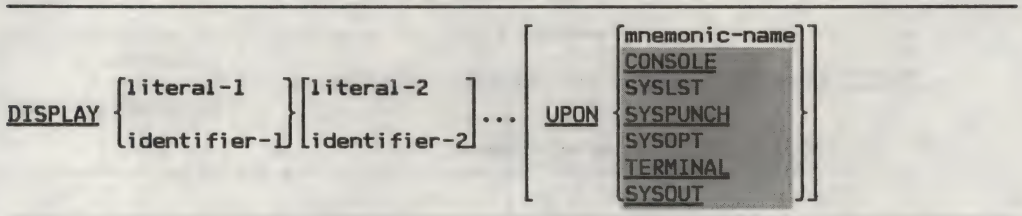
Statement	Calculation
COMPUTE A=(B+C)/D-E.	The value of the expression (B+C)/D-E is assigned to A.
COMPUTE A=2.	The value 2 is assigned to A.

DISPLAY STATEMENT

Function

The DISPLAY statement is used to output small quantities of data.

Format



Rules

1. literal or identifier serves to specify the operands in the order they are to be output. If necessary, the contents of the data item specified by "identifier" are converted to external formats according to the following rules:

   Internal decimal and binary items are converted to external decimal data items. If an item has a sign, an overpunch is generated in the least significant digit.

   Internal floating-point data items are converted to external floating-point data items.

   No other data items require conversion.

   If one of the operands is a figurative constant (except ALL literal), it is output with length 1. If one of the operands is the figurative constant ALL literal, the literal is output once.

2. mnemonic-name must be specified in the SPECIAL-NAMES paragraph and associated with one of the following implementor-names: CONSOLE, SYSLST, SYSLST01-SYSLST99, SYSPUNCH, SYSPCH, SYSOPT, TERMINAL, SYSOUT, job-variable-name (BS2000 job variable).

3. TERMINAL, SYSOUT, CONSOLE, SYSLST, SYSPUNCH, SYSOPT or mnemonic-name specifies the output device where the data is to be output.

   TERMINAL specifies the remote terminal (corresponding to system file SYSOUT at which the user initiated his conversational task). SYSPUNCH specifies the system logical punch device.

   Mnemonic-name specifies a name that was associated with CONSOLE, TERMINAL, SYSOUT, the system logical punch output file (SYSOPT or SYSPUNCH, SYSPCH), the system logical output device (SYSOUT or SYSLST) or a job variable (job-variable-name).

4. If the UPON phrase is omitted, the data is written to the logical output file SYSLST.

Programming considerations

1. A maximum logical record size is assumed for each device. These sizes are listed in Table 2-18.

Device	Maximum record size
CONSOLE	127 characters
SYSLST SYSLST01-SYSLST99	132 characters + 1 control byte
SYSOPT SYSPUNCH SYSPCH	80 characters
SYSOUT TERMINAL	127 characters

Table 2-18 Maximum logical record size for the DISPLAY statement

2. When a DISPLAY statement contains more than one operand, the contents of the specified operands and literals are displayed adjacent to each other, from left to right.

3. If the total character count of all operands is less than the maximum record size, the remainder of the record is padded with blanks. If the total character count of all operands exceeds the maximum record size, additional records are written until all operands are processed.

4. With output to printer, the following entries cause a line feed: DISPLAY, WRITE and WRITE AFTER ADVANCING. A WRITE statement without ADVANCING phrase and a WRITE statement with BEFORE ADVANCING phrase causes the printer to space after printing. Therefore, mixed use of DISPLAY and WRITE statements on the same device within the same program may cause two or more lines to overprint.

5. The maximum record length for job variables is 256 characters. If the total number of characters in the operands exceeds the maximum record length, the record will be truncated to the maximum length.

6. When a job variable is used as a monitoring job variable (MONJV), the system protects the first 128 bytes (system portion) of this job variable against write access. Therefore, only that portion of a data record that begins at position 129 will be written from position 129 of the monitoring job variable. In all other respects, Programming consideration 5 applies for monitoring job variables.

For further information see the "COB1 User's Guide" [7].

Example 2-71

```
SPECIAL-NAMES.
     SYSOUT IS SPECIAL-OUTPUT.
     ...
PROCEDURE DIVISION.
     ...
     DISPLAY OUTPUT-MESSAGE UPON SPECIAL OUTPUT.
```

Here, the mnemonic-name SPECIAL-OUTPUT is associated with the implementor-name SYSOUT in the SPECIAL-NAMES paragraph. The DISPLAY statement writes the current contents of OUTPUT-MESSAGE on SYSOUT.

Example 2-72

```
DISPLAY "LIST OF ITEMS".
```

Since the UPON phrase is omitted, the literal LIST OF ITEMS is displayed to the system logical output device SYSLST.

DIVIDE STATEMENT

Function

The DIVIDE statement is used to divide one numeric operand by another and store the result.

Format 1 of the DIVIDE statement stores the quotient in the dividend field.

Format 2 of the DIVIDE statement stores the quotient in more than one separate result field.

Format 3 of the DIVIDE statement uses the GIVING phrase for storing the quotient and generates the division remainder by means of the REMAINDER phrase.

Format 1

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED] [identifier-3 [ROUNDED]]...
       {literal-1   }

       [ON SIZE ERROR imperative-statement]
```

Rules for format 1

1. Each identifier must refer to an elementary numeric item.
2. The value of identifier-2 is divided by the value of identifier-1 or literal-1. The quotient then replaces the current value of identifier-2. The same applies accordingly to identifier-3... .
3. The maximum size of the quotient after decimal-point alignment is 18 decimal digits.
4. Division by zero always results in overflow (SIZE ERROR).
5. In the case of division with ON SIZE ERROR, it is still possible for a DIVIDE-ERROR to occur since no test is made for quotient overflow (only for division by zero).

For further rules, see under "Options in Arithmetic Expressions"; the ROUNDED, SIZE ERROR and GIVING phrases are described in this section.

Example 2-73

Format 1

Statement	PICTURE of result field	Calculation
DIVIDE A INTO B	9(4)V9(2)	B/A stored as nnnnVnn in B

Format 2

```
DIVIDE {identifier-1} [INTO] {identifier-2}
       {literal-1   }  [BY ]  {literal-2   }

       GIVING identifier-3 [ROUNDED] [identifier-4 [ROUNDED]]...

      [ON SIZE ERROR imperative-statement]
```

Rules for format 2

1. identifier-1 or identifier-2 must refer to an elementary data item.

2. identifier-3 or identifier-4 may refer to an elementary numeric item or to an elementary numeric-edited item.

3. When the INTO phrase is used, the value of identifier-2 or literal-2 is divided by the value of identifier-1 or literal-1; when the BY phrase is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. The quotient is stored in identifier-3, identifier-4,... .

4. The maximum size of the quotient after decimal-point alignment is 18 decimal digits.

5. Division by zero always results in overflow (SIZE ERROR).

6. In the case of division with ON SIZE ERROR, it is still possible for a DIVIDE-ERROR to occur since no test is made for quotient overflow (only for division by zero).

For further rules, see under "Options in Arithmetic Expressions"; the ROUNDED, SIZE ERROR and GIVING phrases are described in this section.

Example 2-74

Format 2

Statement	Result field (C) PICTURE IS:	Calculation
DIVIDE A INTO B GIVING C ROUNDED	S999V99 for C	B/A stored in C as nnnVnn after rounding, if necessary
DIVIDE A BY B, GIVING C, D	9(5) for C 9(4) for D	A/B stored in C as nnnnn, in D as nnnn, after rounding the rightmost character, if necessary.

Format 3

```
DIVIDE {identifier-1} {INTO} {identifier-2}
       {literal-1   } {BY  } {literal-2   }

        GIVING identifier-3 [ROUNDED]
       [REMAINDER identifier-4]

       [ON SIZE ERROR imperative-statement]
```

Rules for format 3

1. identifier-1 or identifier-2 must refer to an elementary numeric data item.

2. identifier-3 or identifier-4 may refer to an elementary numeric data data item or to an elementary numeric-edited data item.

3. When the REMAINDER phrase is used, none of the identifiers may refer to a floating-point elementary item.

4. When INTO is used, the value of identifier-2 or literal-2 is divided by the value of identifier-a or literal-1. When the BY phrase is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. The quotient is stored in identifier-3.

5. When the REMAINDER phrase is used, the remainder of the division is stored in identifier-4.

   The remainder is calculated by subtracting the product of the quotient and the divisor from the dividend.

   If identifier-3 is defined as an elementary numeric-edited item, then the remainder is calculated by using an intermediate field for the quotient, containing the value in an unedited format.

   If both the ROUNDED and the REMAINDER phrases are supplied, then the remainder is calculated by using an intermediate field for the quotient, containing the quotient of the DIVIDE statement in a truncated rather

than rounded format.

6. If the ON SIZE ERROR phrase is specified, and overflow occurs in the quotient, then the remainder will not be calculated. In this case, the contents of the data items referenced by identifier-3 and identifier-4 are therefore unchanged.

   If overflow occurs in the remainder, the value of the data item referenced by identifier-4 is not changed.

7. The precision of the data item required for the REMAINDER phrase (identifier-4) is determined by the calculations described above. Appropriate decimal-point alignment and truncation (rather than rounding) are performed as necessary for the contents of the data item referenced by identifier-4.

8. The maximum size of the quotient after decimal-point alignment is 18 decimal digits.

9. Division by zero always results in overflow (SIZE ERROR).

10. In the case of division with ON SIZE ERROR, it is still possible for a DIVIDE-ERROR to occur since no test is made for quotient overflow (only for division overflow).

For further rules, see under "Options in Arithmetic Statements"; the ROUNDED, SIZE ERROR and GIVING phrases are described in this section.

Example 2-75

Format 3

Statement	Result field (C) PICTURE IS:	Calculation
DIVIDE A INTO B GIVING C ROUNDED	S999V99 for C	B/A stored in C as nnnVnn after rounding the rightmost character, if necessary.
DIVIDE A BY B, GIVING C REMAINDER D	9(5) for C 9(2) for D	A/B stored in C as nnnnn, the remainder, e.g. A-C*B, stored in D as nn.

EXIT STATEMENT

Function

The EXIT statement provides a general exit at the end of a series of procedures.

The EXIT PROGRAM statement, which causes a return from a called program, is discussed under "Program Communication".

Format

```
EXIT
```

Rules

1. The EXIT statement must be preceded by a paragraph-name. It must be the only statement in the paragraph.

2. The EXIT statement is used only to assign a procedure-name to a given point in the program, without affecting the execution of the program.

Programming considerations

1. The EXIT statement, when supplied at the end of a series of procedures, enables the normal execution of that sequence of procedures to be interrupted, passing control directly to the end of the procedure sequence.

2. If control reaches an "EXIT paragraph" and no associated PERFORM or USE statement is active, then control passes to the first sentence of the next paragraph.

Example 2-76

```
          .
          .
          .
          PERFORM X THRU Y.
          .
          .
          .
     X.
          .
          .
          .
          IF A IS ZERO, GO TO Y.
          .
          .
          .
     Y.   EXIT.
          .
          .
          .
```

Here, the "EXIT procedure" is the last procedure covered by the PERFORM statement. If the value of A is zero, then the GO TO statement interrupts the normal flow of execution of the statements ranging from X to Y, and passes control directly to the end of the range of procedures specified by the PERFORM statement. After this, program execution resumes with the next statement after PERFORM.

2

GO TO STATEMENT

Function

The GO TO statement is used to transfer control from one part of the procedure division to another.

Format 1 of the GO TO statement passes control to a specific procedure.

Format 2 of the GO TO statement passes control to one of a series of specified procedures, depending on the value of a data item.

Format 3 of the GO TO statement is used in connection with the ALTER statement (see "ALTER Statement").

Format 1

```
GO TO procedure-name-1
```

Rules for format 1

1. If the GO TO statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement within that sequence.

2. A GO TO statement referenced by an ALTER statement must be preceded by a paragraph-name, and it must be the only statement in this paragraph (see "ALTER Statement").

3. When executing a GO TO statement, control is passed to procedure-name-1. If the GO TO statement was modified by an ALTER statement, control is passed to the procedure-name specified in the last ALTER statement which was executed (see "ALTER Statement").

Example 2-77

Format 1

```
        GO TO END-ROUTINE.
            .
            .
            .
    END-ROUTINE.
        CLOSE CARD-FILE,PRINTER-FILE.
        STOP RUN.
```

In this example, the GO TO statement transfers control to the procedure named END-ROUTINE; the CLOSE statement is executed immediately after the GO TO statement.

Format 2

```
GO TO procedure-name-1 [procedure-name-2]...
      DEPENDING ON identifier
```

Rules for format 2

1. The identifier is the name of an elementary numeric data item described as an integer. The picture-string must indicate four or less digits. The USAGE must be either DISPLAY, COMPUTATIONAL or COMPUTATIONAL-3.

2. When executing a GO TO statement, control is passed to procedure-name-1, procedure-name-2, ... procedure-name-n, depending on the value of the identifier, which may be 1, 2, ... n.
   If the identifier has any value other than 1, 2, ... n, no transfer of control takes place, and processing continues with the next statement within the normal sequence of execution (n represents the specified number of procedure-names).

Programming considerations for format 2

The GO TO statement may specify up to 512 procedure-names.

Example 2-78

Format 2

```
77  A PICTURE 9.
      .
      .
      .
    MOVE 3 TO A.
      .
      .
      .
    GO TO X Y Z DEPENDING ON A.
```

Since the value of A is 3 when the GO TO statement is executed, control is transferred to Z, the third procedure-name in the series.

Format 3

```
GO TO.
```

Rules for format 3

1. The GO TO statement must be a paragraph name, and it must be the only statement in that paragraph.

2. If the GO TO statement was modified by an ALTER statement, control is transferred to the procedure-name specified in the ALTER statement.

Programming consideration for format 3

The GO TO statement must be assigned an entry point by the ALTER statement. Otherwise, the program will terminate with error messages 9040 and 9044 and a dump will be created.

IF STATEMENT

Function

The IF statement causes a condition to be evaluated (see under "Conditions"). The subsequent action of the object program depends upon whether the condition is true or false.

Format

```
IF condition THEN {statement-1...      } ELSE {statement-2...      }
                  {NEXT SENTENCE       }      {NEXT SENTENCE       }
```

Rules

1. statement-1 and statement-2 may consist of one or more imperative statements and/or one conditional statement.

2. The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the closing period of the sentence.

3. If the condition is true, statement-1... is executed. Control then passes to the next sentence, unless statement-1 contains a GO TO statement. If the condition is true and NEXT SENTENCE is supplied rather than statement-1, control passes explicitly to the next sentence of the program.

4. If the condition is false, statement-2... is executed or, if the ELSE phrase is omitted, the next sentence is executed. If the condition is false and NEXT SENTENCE is supplied rather than statement-2, control passes explicitly to the next sentence of the program.

Programming considerations

1. statement-1 and/or statement-2 may contain an IF statement. In this case the IF statement is said to be nested.

   In nested IF statements, processing continues from the concluding period once the innermost IF/ELSE action has been performed.

2. IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not already been paired with an ELSE.

3. In the conditional statement shown in Figure 2-1, C represents "condition", S stands for any number of imperative statements, and the pairing of IF and ELSE is indicated by connecting lines.

The other symbols used in Figure 2-1 indicate the following:

a1 - Statement-1 for IF1 (If C1 is false, the next sentence is executed, since there is no ELSE for it.)

b1 - Statement-1 for IF2

b2 - Statement-2 for IF2

c1 - Statement-1 for IF3

c2 - Statement-2 for IF3

d1 - Statement-1 for IF4 (If C4 is false, the next sentence is executed, since there is no ELSE for it.)

e1 - Statement-1 for IF5

e2 - Statement-2 for IF5

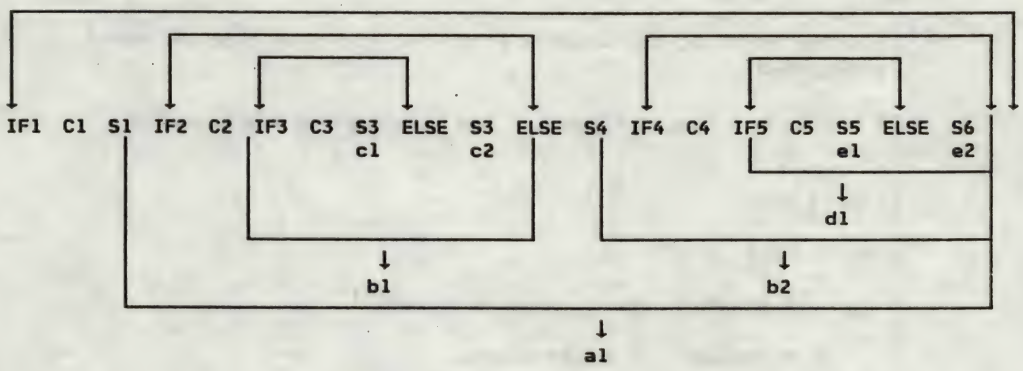


Figure 2-1 Conditional statements with nested IF statements

Example 2-79

```
DDIF A = B statement-1.

Statement-2.
```

If A = B is true, statement-1 and statement-2 are executed.

If A = B is false, only statement-2 is executed.

Example 2-80

```
IF A = B statement-1 ELSE statement-2.

Statement-3.
```

If A = B is true, statement-1 and statement-3 are executed.

If A = B is false, statement-2 and statement-3 are executed.

Example 2-81

```
IF A = B

NEXT SENTENCE

ELSE statement-1

Statement-2.
```

If A = B is true, statement-2 (that is, the next program sentence) is executed.

If A = B is false, statement-1 and statement-2 are executed.

Example 2-82

```
┌─> IF MALE
│
│  ┌─>IF MARRIED, ADD 1 TO MALES-MARRIED
│  │
│  └─> ELSE ADD 1 TO MALES-SINGLE
│
└─> ELSE ADD 1 TO FEMALE.
    Next sentence.
```

This is an example of a nested IF statement. The arrows are used to indicate the IF and ELSE combinations. The second IF is paired with the ELSE immediately following it; and the next ELSE is paired with the last IF that was not previously paired.

2

INITIALIZE STATEMENT

Function

The INITIALIZE statement enables particular categories of data items to be preset with initial values, e.g. zeros for numeric items or blanks for alphanumeric items.

Format

```
INITIALIZE {identifier-1} ...
    [ REPLACING { [ALPHABETIC              ] } DATA BY { [identifier-2] } ... ]
                { [ALPHANUMERIC            ] }         { [          ] }
                { [NUMERIC                 ] }         { [literal-1 ] }
                { [ALPHANUMERIC-EDITED     ] }
                { [NUMERIC-EDITED          ] }
```

Rules

1. literal-1 and identifier-2 represent sending items; identifier-1 is the receiving item.

2. Each data category named in the REPLACING phrase must be allowed as the data category of a receiving item in a MOVE statement in which the sending field is represented by identifier-2 or literal-1.

3. Each data category can only be named once in the REPLACING phrase.

4. The data description entry of identifier-1 or a data item subordinate to identifier-1 must not contain the DEPENDING phrase of the OCCURS clause.

5. An index data item must not occur as the operand of an INITIALIZE statement.

6. The data description entry of identifier-1 must not contain the RENAMES clause.

Programming considerations

1. The keyword following REPLACING identifies a data category (see section 1.4.2.4).

2. Regardless of whether identifier-1 represents an elementary item or a group item, all move operations are performed as though a sequence of MOVE statements had been written, each of them with an elementary item as receiving item.
   The following rules apply:

   a) If identifier-1 is a group, each elementary item in this group is initialized only if it belongs to the data category named in the REPLACING phrase.

b) If identifier-1 represents an elementary item, this elementary item is initialized only if it belongs to the data category named in the REPLACING phrase.

- Initialization is performed as follows:
  Identifier-2 or literal-1 functions as the sending item of an implicit MOVE statement whose receiving item is an elementary item subordinate to identifier-1.

- Except for the cases cited in Programming considerations 3 and 4, all these elementary receiving items are initialized, including all table elements in the group.

3. Index data items and elementary FILLER items are skipped when the INITIALIZE statement is executed.

4. Every data item which is subordinate to a receiving item and whose data description entry contains the REDEFINES clause, or every data item subordinate to an item of this sort, is excluding from the initialization process. The receiving item itself, however, may contain the REDEFINES clause in its data description entry, or it may be subordinated to a data item with a REDEFINES clause.

5. If the REPLACING phrase is omitted, data items of the categories alphabetic, alphanumeric and alphanumeric-edited are initialized with blanks, those of the categories numeric and numeric-edited with zeros.

6. The elementary items represented by identifier-1 are initialized from left to right in the order in which they occur in the INITIALIZE statement. If identifier-1 is a group, the relevant elementary items within this sequence are initialized in the order in which they were defined in the group.

7. If identifier-1 and identifier-2 occupy the same storage space, this statement will produce unpredictable results when executed (see section 2.5.7, "Overlapping Operands").

2

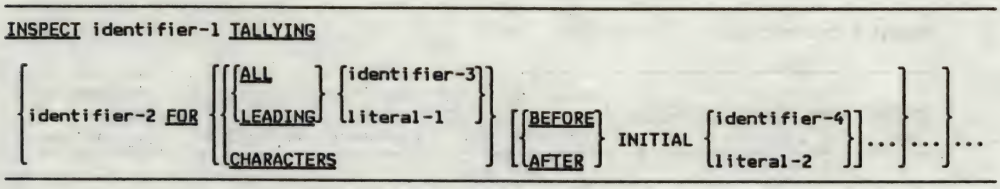
INSPECT STATEMENT

Function

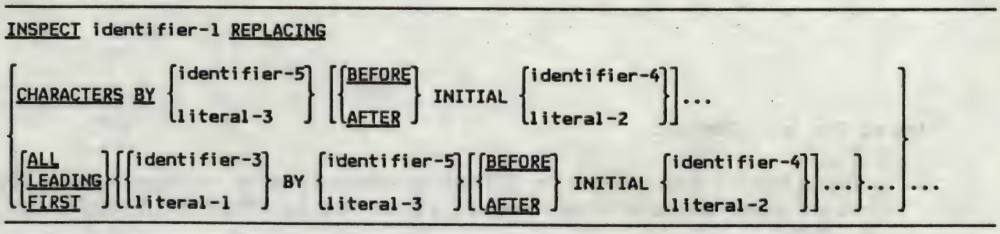
The INSPECT statement enables single characters or groups of characters within a data item to be tallied, replaced or tallied and replaced.

Format 1: tally
Format 2: replace
Format 3: tally and replace
Format 4: convert.

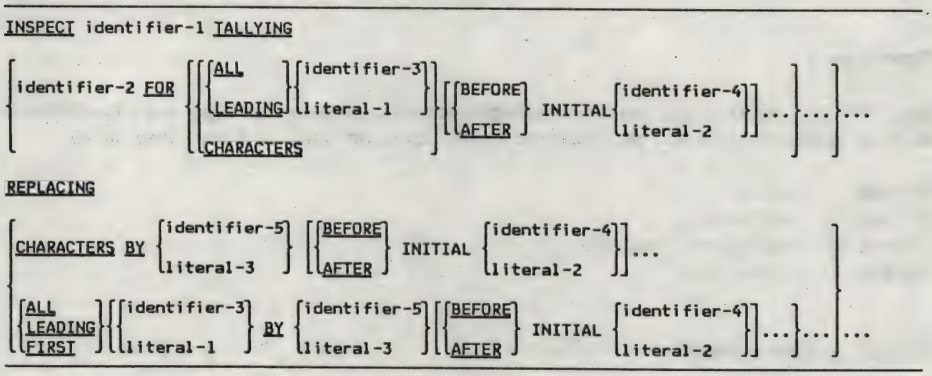
Format 1 (tallying)



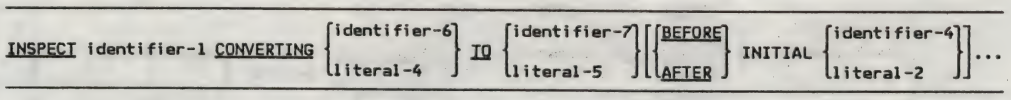
Format 2 (replacing)



Format 3 (tallying and replacing)



Format 4 (converting)



Rules for all formats

1. identifier-1 may reference either a group item or an elementary item, but it must be described (implicitly or explicitly) with USAGE IS DISPLAY.

2. identifier-3, ..., identifier-n must be elementary items implicitly or explicitly described with USAGE IS DISPLAY.

3. literal-1, literal-2, literal-3, literal-4, literal-5 must be either nonnumeric literals or figurative constants which do not begin with ALL.

   If literal-1, literal-2 or literal-4 is a figurative constant, it implicitly represents a single-character item.

4. Only one BEFORE and/or AFTER entry may be assigned to each individual ALL, LEADING, CHARACTERS, FIRST, or CONVERTING phrases.

Rule for format 1 and 3

5. identifier-2 must be an elementary numeric data item.

Rules for format 2 and 3

6. literal-3 or identifier-5 must be equal in size to literal-1 or identifier-3. When a figurative constant is used as literal-3, it is implicitly equal in size to literal-1 or identifier-3.

7. When CHARACTERS is used, literal-2, literal-3, identifier-4 and identifier-5 must each be one character in length.

Rules for format 4

8. literal-5 or identifier-7 must be equal in size to literal-4 or identifier-6.

   When a figurative constant is used as literal-5, it is implicitly equal in size to literal-4 or identifier-6.

9. No character may appear more than once in literal-4 or identifier-6.

Programming considerations for all formats

1. identifier-1 is processed from left to right regardless of its data class.

2. The contents of the data items indicated by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, identifier-7 are treated as follows:

   a) If any of these identifiers is described as alphabetic or alphanumeric, its contents will be treated as a character string.

   b) If any of these identifiers is described as alphanumeric-edited, numeric-edited, or unsigned numeric, it will be treated as though it had been redefined as alphanumeric.

   c) If any of these identifiers is described as signed numeric, it will be treated as though it had been moved to an unsigned numeric data item of the same length (not counting sign position) and this item had then been redefined as alphanumeric (see "MOVE Statement").

      If identifier-1 is described as a signed numeric data item, its original sign will be retained until the INSPECT statement has executed.

3. If any of these identifiers is indexed, the index value for these items will be calculated once only, namely immediately following execution of the INSPECT statement.

4. In Programming considerations 5 to 15, everything that applies to literal-1, literal-2, literal-3, literal-4, or literal-5 applies equally to identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7.

Programming considerations for formats 1, 2 and 3

5. While the contents of identifier-1 are being checked, each occurrence of literal-1 will be tallied (format 1) or all characters matching literal-1 will be replaced by literal-3 (format 2). If CHARACTERS is used, the characters in identifier-1 are tallied or replaced by literal-3 one at a time, depending on where the comparison operation is currently positioned.

6. The TALLYING or REPLACING operands are processed from left to right in the order in which they were specified in the INSPECT statement (comparison cycle). The first comparison cycle starts at the leftmost character in identifier-1.

7. If BEFORE or AFTER are omitted, the comparison operation to determine the occurrences of literal-1 in identifier-1 takes place as follows:

   a) The first literal-1 is compared to a series of contiguous characters within identifier-1 starting with the leftmost character, where the length of this series is equal to the length of literal-1. Only if literal-1 and this portion of identifier-1 are identical, character-for-character, does a match occur.

   b) If no match between literal-1 and identifier-1 occurs, the comparison is repeated with each successive literal-1 until either a match is found or there is no next successive literal-1.

   c) If no match whatsoever occurs between literal-1 and identifier-1, the character position within identifier-1 is shifted one position to the right and the comparison cycle starts again with the first literal-1.

   d) Whenever a match occurs, the comparison cycle is terminated. Identifier-2 is incremented by 1 and/or the characters in identifier-1 which match literal-1 are replaced by literal-3. The character position within identifier-1 is then shifted to the right by the number of characters in literal-1 and the comparison cycle starts again with the first literal-1.

   e) The comparison operation continues until the rightmost character in identifier-1 has either participated successfully in a match or is the character at which a comparison cycle begins.

   f) If ALL is used, points a) to e) apply without restrictions.

      If LEADING is used, the corresponding literal-1 is always involved in the first run-through of the comparison cycle. It only takes part is subsequent comparison cycles if the preceding cycle has produced a match with literal-1.

      If CHARACTERS is used, an implicit single-character operand takes part in the comparison cycle as though it were entered as literal-1. However, no comparison with the contents of identifier-1 takes place; instead, this operand is always considered to match the character in identifier-1 at which the comparison cycle is currently positioned.

8. If BEFORE or AFTER is used, the following restrictions apply to point 7 above:

   a) If BEFORE is used, the operation proceeds as follows:

      literal-1 or (if CHARACTERS has been specified) the implicit operand participates only in those comparison cycles which would have been performed if identifier-1 had ended with the character located immediately in front of the first occurrence of literal-2 within identifier-1.

      If literal-2 does not occur at all within identifier-1, the comparison proceeds as if BEFORE had never been entered.

   b) If AFTER is used, the same considerations apply as in a). That is, a search is made in identifier-1 for literal-2. If literal-2 is located, the record pointer is positioned to the character within identifier-1 which is located immediately to the right of literal-2. From this point on, literal-1 or (if CHARACTERS has been specified) the implicit operand participates in the subsequent comparison cycles.

      If literal-2 does not occur at all within identifier-1, literal-1 or (if CHARACTERS has been specified) the implicit operand is not involved in a comparison cycle.

Programming considerations for format 1 and 3

9. The contents of identifier-2 are not initialized when the INSPECT statement is executed.

10. If identifier-1, identifier-3 or identifier-4 occupies the same memory area as identifier-2, the results of the INSPECT statement will be unpredictable, even when these identifiers are defined in the same data description entry (see "Overlapping Operands").

Programming considerations for format 2 and 3

11. The mandatory words ALL, LEADING and FIRST are adjectives which apply to all subsequent BY phrases until the next adjective is entered.

12. If FIRST is used, literal-1 will be replaced by literal-3 within identifier-1 only at the position where it occurs for the first time. This rule applies to all successive FIRST phrases, regardless of the value of literal-1.

13. If identifier-3, identifier-4 or identifier-5 occupies the same memory area as identifier-1, the results of the INSPECT statement will be unpredictable, even when these identifiers are described in the same data description entry (see "Overlapping Operands").

Programming consideration for format 3

14. A format-3 INSPECT statement is executed as though it were two successive INSPECT statements referring to the same identifier-1, namely one format-1 INSPECT statement (with TALLYING phrase) and a format-2 INSPECT statement (with REPLACING phrase).
    The rules given for formats 1 and 2 apply accordingly.

Programming considerations for format 4

15. A format-4 INSPECT statement is interpreted and executed as though it were a format-2 INSPECT statement containing a series of ALL phrases (one for each character in literal-4 or identifier-6) which refer to the same identifier-1.

    Thus, each character in literal-4 or identifier-6 and the corresponding character in literal-5 or identifier-7 is interpreted as though it were a self-contained literal-1 (or identifier-3) or literal-3 (identifier-5) in format 2.

    The unique assignment of characters from literal-4 (identifier-6) and literal-5 (identifier-7) results from their position within the data item.

16. If identifier-4, identifier-6 or identifier-7 occupies the same memory area as identifier-1, the results of the INSPECT statement will be unpredictable, even when the identifiers are defined in the same data description entry (see "Overlapping Operands").

Example 2-83 (all formats)

```
ID DIVISION.
PROGRAM-ID.  INSPECTT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COUNTER1  PIC 99    VALUE ZEROES.
01 COUNTER2  PIC 99    VALUE 0.
01 COUNTER3  PIC 99    VALUE 0.
01 FIELD     PIC X(20) VALUE SPACES.
PROCEDURE DIVISION.
BEG.
    MOVE "BBYZYZBBYZAXBXBBX" TO FIELD.
    INSPECT FIELD TALLYING
       COUNTER1 FOR ALL "X" AFTER INITIAL "A"
       COUNTER2 FOR LEADING "YZ" AFTER INITIAL "BB"          (1)
       COUNTER3 FOR CHARACTERS BEFORE INITIAL "A".
    DISPLAY "AFTER INSPECT".
    DISPLAY "COUNTER1 = *" COUNTER1 "*".
    DISPLAY "COUNTER2 = *" COUNTER2 "*".
    DISPLAY "COUNTER3 = *" COUNTER3 "*".
*
    MOVE "MR. COBOLUSER" TO FIELD.
    DISPLAY "BEFORE INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
    INSPECT FIELD REPLACING
       CHARACTERS BY "X" AFTER INITIAL "MR. "
                         BEFORE INITIAL "U".
    DISPLAY "AFTER INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
*
    MOVE "ALGOL-PROGRAM" TO FIELD.
    DISPLAY "BEFORE INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
    INSPECT FIELD REPLACING
       ALL "A" BY "C" BEFORE INITIAL "P"
       ALL "L" BY "O" BEFORE INITIAL "G"
       ALL "G" BY "B" BEFORE INITIAL "P".
    DISPLAY "AFTER INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
*
    MOVE "XXYZYZXXYZ-XYZXYZ" TO FIELD.
    DISPLAY "BEFORE INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
    INSPECT FIELD REPLACING
       LEADING "YZ" BY "AB" BEFORE INITIAL "-"
                            AFTER INITIAL "XX"               (2)
       FIRST "YZ" BY "CD" AFTER INITIAL "-".
    DISPLAY "AFTER INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
*
    MOVE "CE#CGDHDEF-CD#F" TO FIELD.
    DISPLAY "BEFORE INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
    INSPECT FIELD CONVERTING                                 (3)
       "CDEF" BY "UVWU" AFTER "#"
                        BEFORE "-".
    DISPLAY "AFTER INSPECT".
    DISPLAY "FIELD = *" FIELD "*".
*END.
    STOP RUN.
```

Result:

AFTER INSPECT (1)

```
COUNTER1 = *03*
COUNTER2 = *02*
COUNTER3 = *06*
```

BEFORE INSPECT		AFTER INSPECT		
FIELD = *MR. COBOLUSER	*	FIELD = *MR. XXXXXUSER	*	
FIELD = *ALGOL-PROGRAM	*	FIELD = *COBOL-PROGRAM	*	(2)
FIELD = *XXYZYZXXYZ-XYZXYZ	*	FIELD = *XXABABXXYZ-XCDXYZ	*	
FIELD = *CE#CGDHDEF-CD#F	*	FIELD = *CE#UGVHVWU-CD#F	*	(3)

Explanations

(1) In the case of COUNTER2, the instances of YZ underlined in the following string were tallied in FIELD:
BBYZYZBBYZAXBXBBX

In other words, with each of the underlined YZ pairs there was a match in the sense of rule 7f. Hence, on both occasions the compare cycle resumed with the first literal-1, i.e. with COUNTER1.

The result is as follows:
COUNTER3 is not incremented for the underlined YZ pairs. Hence, only the following underlined characters are tallied:
BBYZYZBBYZAXBXBBX

Thus, COUNTER3 is equal to 6.

COUNTER3 would be equal to 10 following an INSPECT statement in which COUNTER3 was the first or the only literal-1.

(2) The replacement of leading YZ pairs by AB is caused by entering AFTER INITIAL "XX". BEFORE INITIAL "-" has no effect since no leading YZ pairs are present from the outset.

(3) This INSPECT statement has the same effect as the following statement:

```
INSPECT FIELD REPLACING
    ALL "C" BY "U" AFTER "#" BEFORE "-"
    ALL "D" BY "V" AFTER "#" BEFORE "-"
    ALL "E" BY "W" AFTER "#" BEFORE "-"
    ALL "F" BY "U" AFTER "#" BEFORE "-".
```

2

MOVE STATEMENT

Function

The MOVE statement transfers data from one data item to one or more other data items.

Format 1 of the MOVE statement moves one data item to one or more other data items.

Format 2 of the MOVE statement moves corresponding data items from one group to another.

Format 1

```
MOVE {identifier-1} TO identifier-2 [identifier-3]...
     {literal     }
```

Rules for format 1

1. identifier-1 or literal represents the sending field. The data in this field is moved to the receiving field specified by identifier-2.

   The same data is also moved to each additional receiving field if identifier-3 ... is supplied in the statement.

The following rules specified for identifier-2 are applicable to identifier-3 ... also.

2. An index data item cannot appear as an operand in a MOVE statement.

3. Any subscripting or indexing in conjunction with identifier-2, ... is evaluated just before the data is transferred to the relevant data item.

4. Any required conversion of the data from one form of internal representation to another takes place during valid moves, as well as any required editing of the receiving field. This is described more fully in the "Programming considerations" which follow.

Programming considerations

See "Programming considerations for both formats".

Format 2

```
MOVE {CORRESPONDING} identifier-1 TO identifier-2...
     {CORR          }
```

Rules for format 2

The following rules for identifier-2 also apply to any additional identifiers which follow identifier-2.

1. CORR is the abbreviation for CORRESPONDING.

2. identifier-1 specifies a group item which contains the elementary data items to be moved.

3. identifier-2 specifies a group item which contains the receiving fields for the move.

4. The data items selected from the first operand (identifier-1) are moved to corresponding data items within the second operand (identifier-2). Data items from each group are considered to be corresponding when both data items have the same name and qualification up to, but not necessarily including, identifier-1 and identifier-2. The results of the MOVE CORRESPONDING statement are the same as if the user had specified each pair of corresponding identifiers in a separate MOVE statement of format 1 (for further rules see the CORRESPONDING phrase).

5. An index data item cannot appear as an operand in a MOVE statement.

6. Any subscripting or indexing in conjunction with identifier-2, ... is evaluated just before the data is transferred to the relevant data item.

7. Any required conversion of the data from one form of internal representation to another takes place during valid moves, as well as any required editing of the receiving field. This is described more fully in the "Programming considerations" which follow.

Programming considerations for format 2

1. At least one of the data items in each pair of corresponding data items must be an elementary data item (note that the MOVE statement differs from arithmetic statements in this respect: in arithmetic statements using the CORRESPONDING phrase, both corresponding items must be elementary data items).

2. A data item that is subordinate to identifier-1 or identifier-2 and contains an OCCURS, REDEFINES, USAGE IS INDEX or RENAMES clause will be ignored. However, identifier-1 or identifier-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

For further **programming considerations** see "Programming considerations for both formats".

Example 2-84 (format 2)

Procedure Division statement:

```
MOVE CORRESPONDING EMPLOYEE-RECORD TO PAYROLL-CHECK.
```

Data Division entries:

```
01  EMPLOYEE-RECORD.
    02  EMPLOYEE-NUMBER.
        03  PLANT-LOCATION...
        03  CLOCK-NUMBER.
            04 SHIFT-CODE...
            04 CONTROL-NUMBER...
    02  INCOME.
        03  HOURS-WORKED...
        03  PAY-RATE...
    02  FICA-RATE...
    02  DEDUCTIONS...
```

```
01  PAYROLL-CHECK.
    02  EMPLOYEE-NUMBER.
        03  CLOCK-NUMBER...
        03  FILLER...
    02  DEDUCTIONS.
        03  FICA-RATE...
        03  WITHHOLDING-TAX...
        03  PERSONAL-LOANS...
    02  INCOME.
        03  HOURS-WORKED...
        03  PAY-RATE...
    02  NET-PAY...
    02  EMPLOYEE-NAME.
        03  SHIFT-CODE...
```

According to the MOVE CORRESPONDING rules, the following data items would be moved:

Sending area	Receiving area
CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	CLOCK-NUMBER OF EMPLOYEE-NUMBER OF PAYROLL-CHECK
HOURS-WORKED OF INCOME OF EMPLOYEE-RECORD	HOURS-WORKED OF INCOME OF PAYROLL-CHECK
PAY-RATE OF INCOME OF EMPLOYEE-RECORD	PAY-RATE OF INCOME OF PAYROLL-CHECK
DEDUCTIONS OF EMPLOYEE-RECORD	DEDUCTIONS OF PAYROLL-CHECK

The following items are not moved, for the reasons stated:

Field	Reason
EMPLOYEE-NUMBER	Item is not elementary in either group.
PLANT-LOCATION OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Item does not appear in PAYROLL-CHECK.
SHIFT-CODE OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK.
CONTROL-NUMBER OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Item does not appear in PAYROLL-CHECK.
INCOME	Item is not elementary in either group.
FICA-RATE OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK.

Programming considerations for both formats

1. Any move in which the sending and receiving items are both elementary items is an **elementary move**. Table 2-19 lists the classes to which elementary items belong.

Item	Category
Elementary data item	Numeric, alphabetic or alphanumeric, alphanumeric edited, numeric edited
Numeric literal	Numeric
Nonnumeric literal	Alphanumeric

Table 2-19 Categories of elementary items

Any move in which either or both items are group items is a **group move**.

2. Table 2-20 lists all elementary and group moves, and indicates which moves are valid. The remaining programming considerations describe the execution of valid moves.

Sending field	Receiving field									
	GR	AL	AN	ED	BI	NE	ANE	ID	EF	IF
Group (GR)	Y	Y	Y	Y1	Y1	Y1	Y1	Y1	Y1	Y1
Alphabetic (AL)	Y	Y	Y	N	N	N	Y	N	N	N
Alphanumeric (AN)	Y	Y	Y	Y4	Y4	Y4	Y	Y4	Y4	Y4
External decimal (ED)	Y1	N	Y2	Y	Y	Y	Y2	Y	Y	Y
Binary (BI)	Y1	N	Y2	Y	Y	Y	Y2	Y	Y	Y
Numeric edited (ED)	Y	N	Y	N	N	N	Y	N	N	N
Alphanumeric edited (ANE)	Y	Y	Y	N	N	N	Y	N	N	N
ZEROS (numeric or alphanumeric)	Y	N	Y	Y3	Y3	Y3	Y	Y3	Y3	Y3
SPACES (AN)	Y	Y	Y	N	N	N	Y	N	N	N
HIGH-VALUE, LOW-VALUE or QUOTES	Y	N	Y	N	N	N	Y	N	N	N
ALL literal	Y	Y	Y	Y5	Y5	Y5	Y	Y5	N	N
Numeric literal	Y1	Y	Y	Y5	Y5	Y5	Y	Y5	N	N
Nonnumeric literal	Y	Y	Y	Y5	Y5	Y5	Y	Y5	N	N
Internal decimal (ID)	Y1	N	Y2	Y	Y	Y	Y2	Y	Y	Y
Ext. fl. point (EF)	Y1	N	N	Y	Y	Y	N	Y	Y	Y
Int. fl. point (IF)	Y1	N	N	Y	Y	Y	N	Y	Y	Y
Fl. point literal	Y1	N	N	Y	Y	Y	N	Y	Y	Y

Table 2-20 Permissible elementary and group moves

Y denotes valid move, and N denotes invalid move
Y1 Move without conversion (like AN to AN)
Y2 Only if the decimal point is in the rightmost position (to the right of the least significant digit)
Y3 Numeric move
Y4 The alphanumeric field is treated as an ED (integer) item (may only contain numeric characters)
Y5 The literal must consist only of numeric characters

3. Any editing specified for the receiving data item takes place during an elementary move (some examples are given below).

4. There are two types of elementary moves: alphanumeric moves and numeric moves.

   a) An **alphanumeric move** takes place when an alphanumeric-edited, alphanumeric, or alphabetic item is the receiving item. Table 6-12 lists the rules for this type of move.

Rule	Sending field		Receiving field	
	PICTURE IS	Value	PICTURE IS	Value
The characters are placed in the receiving field from left to right, unless the receiving field is specified as JUSTIFIED RIGHT. If the receiving field is not completely filled by the data being moved, the remaining positions are filled with spaces.	XXX	M8N	XXXXX	M8N__
If the sending field is longer than the receiving field, the move is terminated, as soon as the receiving field is filled. Excess characters are truncated. If the receiving field is described as JUSTIFIED RIGHT, truncation takes places on the left.	AAAAAA	XYZ123	AAA AAA JUST RIGHT	XYZ 123
If the sending field has an operational sign, its absolute value is moved.	S999	-333	XXX	333

Table 2-21 Rules for alphanumeric moves

b) A **numeric move** takes place when a numeric item is moved to a numeric item or numeric-edited item. Table 2-22 lists the rules for this type of move.

Rule	Sending field 1)		Receiving field 1)	
	PICTURE IS	Value	PICTURE IS	Value
The items are aligned on their decimal points. If the sending field is larger than the receiving field, truncation may occur at either end. If the receiving field is larger than the sending field, unused character positions are either zero filled; or they are replaced as specified by editing characters.	9V999	8V765	V99	V76
	9V9	1V2	99V99	01V20
	99V99	67V89	$$$99.99	␣␣$67.89
If the sending field has an operational sign and the receiving field has no operational sign, then the absolute value of the sending field is moved.	S999	-333	9999	0333
If no assumed or actual decimal points are specified, data is right justified in the receiving field.	99	15	999	015

Table 2-22 Rules for numeric moves

1) V is the assumed decimal point. Any necessary conversion of data from one form of internal representation to another takes place during the move.

5. Any move involving group items is treated exactly as though it were an alphanumeric elementary move.

   If a group item contains a subordinate item whose description includes an OCCURS...DEPENDING ON clause, then the group item is said to be of variable length.

   If such a variable-length group item is involved in a move, only the current active length of the group item is used in the move. The length is determined by the DEPENDING ON phrase. This applies to both sending and receiving field.

6. If the sending or receiving operands of a MOVE statement share the same storage area (i.e. if the operands "overlap"), then execution of that statement produces unpredictable results.

MULTIPLY STATEMENT

Function

The MULTIPLY statement is used to perform multiplication of two numeric operands and store the result.

Format 1 of the MULTIPLY statement stores the products in the specified multiplier.

Format 2 of the MULTIPLY statement uses the GIVING phrase.

Format 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED] [identifier-3 [ROUNDED]]...
         { literal-1    }

         [ON SIZE ERROR imperative-statement]
```

Rules for format 1

1. Each identifier must reference an elementary numeric item.
2. The value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The multiplication product replaces the current value of identifier-2 (the same applies to identifier-3,...).
3. The maximum size of the product after decimal-point alignment is 18 decimal digits.

For further rules, see under "Options in Arithmetic Statements" ; the ROUNDED, SIZE ERROR and GIVING phrases are discussed in this chapter.

Example 2-85 (format 1)

Statement	PICTURE of result field	Calculation
MULTIPLY A BY B	999	A*B stored in B as nnn

Format 2

```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }   { literal-2    }

         GIVING identifier-3 [ROUNDED] [identifier-4 [ROUNDED]]...
         [ON SIZE ERROR imperative-statement]
```

Rules for format 2

1. Each identifier preceding the word GIVING must reference an elementary numeric item.

2. identifier-3, identifier-4,... may reference an elementary numeric item or an elementary numeric-edited item.

3. The value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2, and the product is stored in identifier-3 (the same applies to identifier-4, ...).

4. The maximum size of the product after decimal-point alignment is 18 decimal digits.

For further rules, see under "Options in Arithmetic Statements"; the ROUNDED, SIZE ERROR and GIVING phrases are discussed in this chapter.

Example 2-86 (format 2)

Statement	PICTURE of result field (C)	Calculation
MULTIPLY A BY B GIVING C	9(5)	A*B stored C as nnnnn

PERFORM STATEMENT

Function

The PERFORM statement is used to depart from the normal sequence of procedures, in order to execute one or more procedures either a specified number of times or until a specified condition is satisfied. After the procedures have been executed, the PERFORM statement returns control to the normal sequence unless a GO TO, EXIT PROGRAM or STOP RUN statement is issued.

Format 1 of the PERFORM statement executes a specified group of procedures one time.

Format 2 of the PERFORM statement executes a group of procedures a specified number of times.

Format 3 of the PERFORM statement executes a group of procedures until a specified condition is true.

Format 4 of the PERFORM statement changes the values of one or more identifiers or index names in ascending or descending order, and executes a group of procedures one or more times, based on this action.

Format 1

```
PERFORM procedure-name-1 [ { THRU      } procedure-name-2 ]
                         [ { THROUGH   }                  ]
```

Rules for format 1

1. The statements within the range of the PERFORM statement are executed once, and control returns to the statement following the PERFORM statement.

2. When both procedure-name-1 and procedure-name-2 are specified, and each is the name of a procedure within the declarative section, then both procedure-names must be within the same declarative section.

3. When executing a PERFORM statement, control is passed to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made in this manner:

   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.

   b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

   c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

4. During program execution, if control passes to procedure-name-1 and procedure-name-2 by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

5. If two paragraphs in different sections have the same name, the paragraph-name in the current section is chosen, i.e. the section in which the PERFORM statement is located.

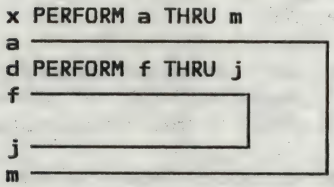
Programming considerations for format 1

1. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.

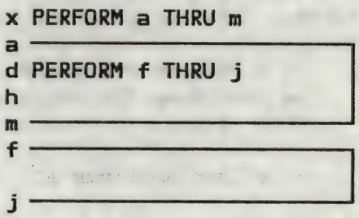
2. When only procedure-name-1 is specified, GO TO and PERFORM statements may occur within the procedure.

3. If a procedure within the range of a PERFORM statement includes another PERFORM statement, the range of the embedded PERFORM statement must be either totally contained within or totally removed from the range of the initial PERFORM statement, as shown in the examples which follow:

Correct:

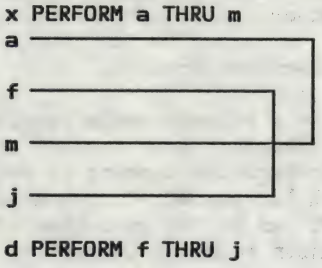


Correct:

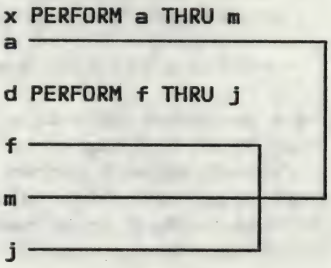


The range of procedures associated with a PERFORM statement may overlap or intersect the range associated with another PERFORM statement, if one PERFORM statement is not contained within the range of the other PERFORM statement. This is illustrated below:

Correct:



Incorrect:

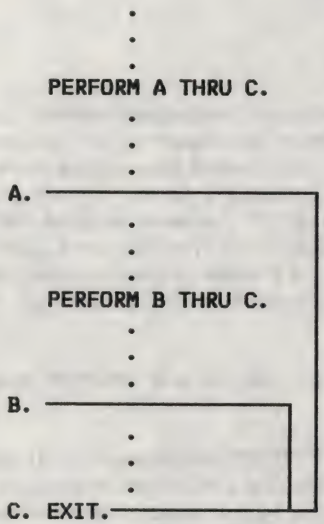


4. If the range of a PERFORM statement starts immediately after the PERFORM statement, the procedures within that range will be executed one extra time.

5. Nested PERFORM statements can have the same exit as the outer PERFORM statement.

   This exit must be the name of a paragraph consisting solely of the statement EXIT.

Example 2-87



For further information see the "COBOL User's Guide" [7].

6. The following information on segmentation is relevant here (for further details, refer to "Segmentation"):

   a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may, in its range, contain only the following procedures:

      - either sections all with a segment number less than 50, or

      - sections which, as a whole, are contained in one single segment whose segment number is greater than 49.

   b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may, in its range, contain only the following procedures:

      - either sections all with the same segment number as the section containing the PERFORM statement or

      - sections all with a segment number less than 50.

   c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 2-88 (format 1)

```
PERFORM X.
ADD A TO B.
```

Let X be a paragraph-name. In this case, all statements in the paragraph named X are executed, and control is then returned to the ADD statement following the PERFORM statement.

Example 2-89

```
          .
          .
          .
     PERFORM X1 THRU X3.
          .
          .
          .
X SECTION.
X1.
          .
          .
          .
X2.
          .
          .
          .
X3.
          .
          .
          .
Y SECTION.
          .
          .
          .
```

The PERFORM statement causes all statements in the paragraphs named X1, X2 and X3 to be executed.

Example 2-90

The same effect would result from the execution of the statement:

```
PERFORM X.
```

Format 2

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ] [ { identifier-1 } ] TIMES
                         [ { THROUGH }                  ] [ { integer-1    } ]
```

Rules for format 2

1. identifier-1 specifies an elementary numeric data item described as an integer.

2. The statements within the range of the PERFORM statement are executed the number of times specified by integer-1 or by the initial value of identifier-1. Control is then returned to the next executable statement following the PERFORM statement.

3. If the value of identifier-1 is zero or negative when the PERFORM statement is to be executed, control immediately passes to the next executable statement following the PERFORM statement.

4. When both procedure-name-1 and procedure-name-2 are specified, and each is the name of a procedure within the declarative section, then both procedure-names must be within the same declarative section.

5. When executing a PERFORM statement, control is passed to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made in this manner.

   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.

   b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

   c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

   d) If procedure-name-2 is specified and it is a section name, then the return is after the last statement of the last paragraph in the section.

6. During program execution, if control passes to procedure-name-1 and procedure-name-2 by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

7. If two paragraphs in different sections have the same name, the paragraph-name in the current section is chosen, i.e. the section in which the PERFORM statement is located.

Programming considerations for format 2

1. If the identifier-1 is changed during the execution of the PERFORM statement, the change has no effect on the number of times the specified statements are performed.

2

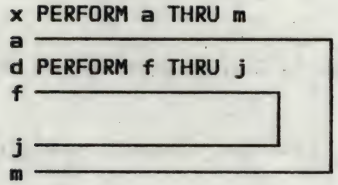
2. The value of identifier-1 or integer-1 must not exceed 32,767.

3. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of paragraph consisting of the EXIT statement to which all of these paths must lead.

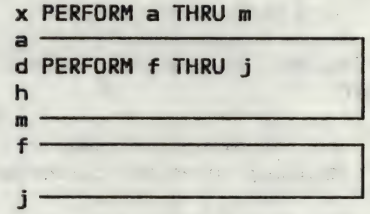
4. When only procedure-name-1 is specified, GO TO and PERFORM statements may occur within the procedure.

5. When a procedure within the range of a PERFORM statement includes another PERFORM statement, the range of the embedded PERFORM statement must be either totally contained within or totally removed from the range of the initial PERFORM statement:

Correct:

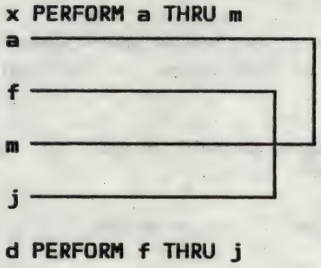


Correct:

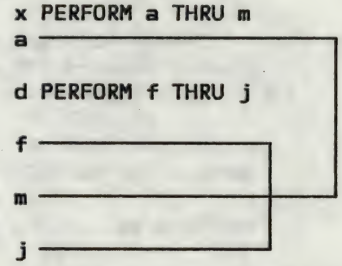


The range of procedures associated with a PERFORM statement may overlap or intersect the range associated with another PERFORM statement, if one PERFORM statement is not contained within the range of the other PERFORM statement. This is illustrated below:

Correct:



Incorrect:

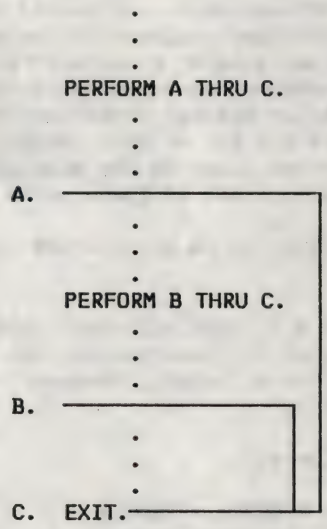


6. If the range of a PERFORM statement starts immediately after the PERFORM statement, the procedures within that range will be executed one extra time.

7. Nested PERFORM statements can have the same exit as the outer PERFORM statement.

   This exit must be the name of a paragraph consisting solely of the statement EXIT.

Example 2-91



For further information see the "COB1 User's Guide" [7], COBRUN Operand NESTPF.

8. The following information on segmentation is relevant here (for further details; refer to "Segmentation"):

   a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may, it its range, contain only the following procedures:

      - either sections all with a segment number less than 50, or

      - sections which, as a whole, are contained in one single segment whose segment number is greater than 49.

   b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may, in its range, contain only the following procedures:

      - either sections all with the same segment number as the section containing the PERFORM statement or

      - sections all with a segment number less than 50.

   c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 2-92 (format 2)

```
PERFORM X 5 TIMES.
```

All statements in the paragraph named X are performed five times. Control then passes to the statement following the PERFORM statement.

Example 2-93 (format 2)

```
          .
          .
          .
    77 A PICTURE 9.
          .
          .
          .
        MOVE 3 TO A.
          .
          .
          .
        PERFORM X A TIMES.
          .
          .
          .
    X.
          .
          .
          .
        ADD 1 TO A.
    Y.
          .
          .
          .
```

Since the value of A is 3 when the PERFORM statement is executed, the paragraph named X is executed three times.

The reference to A within X has no effect on the PERFORM statement.

Format 3

```
PERFORM procedure-name-1 [ { THRU      } procedure-name-2 ] UNTIL condition-1
                         [ { THROUGH   }                  ]
```

Rules for format 3

1. The statements within the range of the PERFORM statement are executed until the condition is true. When the condition is true, control is transferred to the next executable statement following the PERFORM statement.

2. When both procedure-name1 and procedure-name-2 are specified, and each is the name of a procedure within the declarative section, then both procedure-names must be within the same declarative section.

3. When executing a PERFORM statement, control is passed to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made in this manner:

   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.

   b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

   c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

   d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

4. During program execution, if control passes to procedure-name-1 and procedure-name-2 by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

5. If two paragraphs in different sections have the same name, the paragraph-name in the current section is chosen, i.e. the section in which the PERFORM statement is located.

Programming consideration for format 3

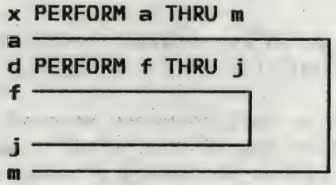
1. If the condition is true when the PERFORM statement is entered, control is not transferred to procedure-name-1; instead, control is transferred immediately to the next executable statement following the PERFORM statement.

2. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.

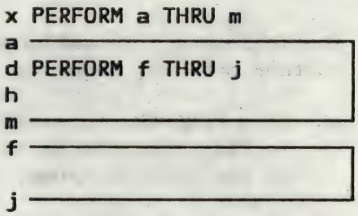
3. When only procedure-name-1 is specified, GO TO and PERFORM statements may occur within the procedure.

4. When a procedure within the range of a PERFORM statement includes another PERFORM statement, the range of the embedded PERFORM statement must be either totally contained within or totally removed from the range of the initial PERFORM statement:

Correct:

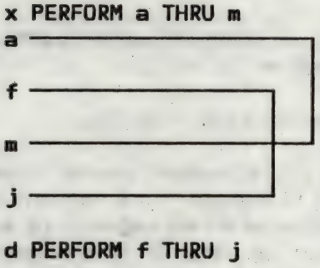


Correct:

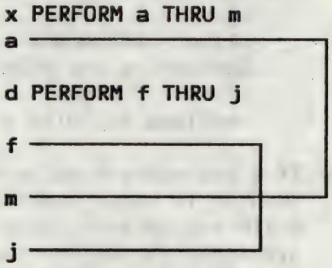


The range of procedures associated with a PERFORM statement may overlap or intersect the range associated with another PERFORM statement, if one PERFORM statement is not contained within the range of the other PERFORM statement. This is illustrated below:

Correct:



Incorrect:

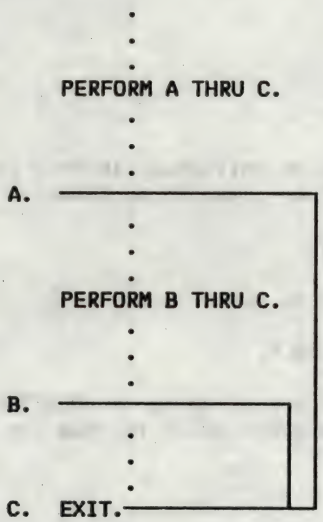


5. If the range of a PERFORM statement starts immediately after the PERFORM statement, the procedures within that range will be executed one extra time.

6. Nested PERFORM statement can have the same exit as the outer PERFORM statement.

   This exit must be the name of a paragraph consisting solely of the statement EXIT.

Example 2-94



For further information see the "COBOL User's Guide" [7], COBRUN operand NESTPF.

7. The following information on segmentation is relevant here (for further details, refer to "Segmentation"):

   a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may, in its range, contain only the following procedures:

      - either sections all with a segment number less than 50, or

      - sections which, as a whole, are contained in one single segment whose segment number is greater than 49.

   b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may, in its range, contain only the following procedures:

      - either sections all with the same segment number as the section containing the PERFORM statement or

      - sections all with a segment number less than 50.

   c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 2-95 (format 3)

```
        PERFORM X UNTIL A GREATER THAN 3.
        .
        .
        .
X.
        .
        .
        .
        COMPUTE A = A + 1.
        .
        .
        .
Y.
```

Assume A = 1, when the PERFORM statement is initiated. In this case, the statements in the paragraph named X are performed three times:

The first time X is performed, A is set to 2.

The second time X is performed, A is set to 3.

The third time X is performed, A is set to 4.

Since 4 is greater than 3, the condition specified in the PERFORM statement is true. Thus, control passes to the statement which follows the PERFORM statement.

Format 4

```
PERFORM procedure-name-1 [[THRU      ] procedure-name-2]
                         [{       }]
                         [{THROUGH}]

VARYING {     index-1   } FROM {index-2   } BY {literal-3   } UNTIL condition-1
        {identifier-1   }      {literal-2   }    {identifier-3}
                               {identifier-2}

[AFTER  {index-4   } FROM {index-5   } BY {literal-6   } UNTIL condition-2
        {identifier-4}    {literal-5   }    {identifier-6}
                          {identifier-5}

[AFTER  {index-7   } FROM {index-8   } BY {literal-9   } UNTIL condition-3]]
        {identifier-7}    {literal-8   }    {identifier-9}
                          {identifier-8}
```

Rules for format 4

1. Each identifier represents a numeric item described as an integer. However, the compiler allows each identifier to be described as a nonintegral numeric item.

2. Each literal must be numeric.

3. The literals used in the BY phrase must not be zero.

4. The values of identifiers, index-names, or literals in the associated FROM phrase must be positive. However, the compiler allows initial values to be zero or negative.

5. This variant of the PERFORM statement is used to augment or decrement the values of one or more identifiers or index-names in a specific manner while the PERFORM statement is executing. Execution depends on the number of identifiers or index-names that are varied. The following rules describe what happens when one, two and three identifiers or index-names are varied.

   In this description, every reference to an identifier is also applicable to index-name, except when the specified identifier occurs in a BY phrase. Also, when index-names are used, the FROM and BY phrases have the same effect as SET statements (see "Table Handling").

6. When **one** identifier is varied, the following sequence of events takes place:

   a) identifier-1 is set to its initial value (the value of identifier-2 or literal-2).

   b) The condition of the UNTIL phrase is tested.

      - If the condition is true, control passes to the statement following the PERFORM statement (in this case, steps c and d are not performed).

      - If the condition is false, the procedures in the range of the PERFORM statement are executed once.

c) The value of identifier-1 is changed by the specified increment or decrement value (the value of identifier-3 or literal-3).

d) Steps b and c are repeated as required:
The specified procedures are executed, and the identifier changed, until the condition is true.

7. When **two** identifiers are varied, the following sequence of events takes place:

a) identifier-1 and identifier-4 are set to their initial values (the value of identifier-2 or literal-2, or the value of identifier-5 or literal-5, respectively).

b) condition-1 is tested.

If condition-1 is true, control passes to the statement following the PERFORM statement. In this case, steps c-h are not performed.

If condition-1 is false, step c is performed.

c) condition-2 is tested.

If condition-2 is true, steps g and h are performed.

If condition-2 is false, steps d-f are performed.

d) The procedures within the range of the PERFORM statement are executed.

e) The value of identifier-4 is changed by the specified increment or decrement (i.e. by the value of identifier-6 or literal-6).

f) Step c is repeated; and steps d and e are repeated, if necessary. In other words, the specified procedures are performed and identifier-4 is varied, until condition-2 is true.

g) identifier-4 is reset to its initial value, and the value of identifier-1 is changed by the specified increment or decrement (i.e. by the value of identifier-3 or literal-3).

h) Step b is repeated. If necessary, steps c-g are repeated. In other words, condition-2 is tested and the subsequent actions described above are taken, until condition-1 is true.

When **three** identifiers are varied, the mechanism is similar to that for two identifiers. In this case, identifier-7 goes through a complete cycle each time that identifier-4 is varied; and identifier-4 goes through a complete cycle each time that identifier-1 is varied (see Example 4-1 as an example of a PERFORM statement with three identifiers to be varied).

8. When both procedure-name-1 and procedure-name-2 are specified, and each is the name of a procedure within the declarative section, then both procedure-names must be within the same declarative section.

9. When executing a PERFORM statement, control is passed to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made in this manner:

a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.

b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

10. During program execution, if control passes to procedure-name-1 and procedure-name-2 by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

11. If two paragraphs in different sections have the same name, the paragraph-name in the current section is chosen, i.e. the section in which the PERFORM statement is located.

Programming considerations for format 4

1. At the completion of the PERFORM statement, identifier-1, identifier-4, and identifier-7 have the following values:

   If condition-1 was true at the beginning of the PERFORM statement, then identifier-1 contains the current value of identifier-3 while identifier-4 and identifier-7 retain their initial values. Otherwise, identifier-1 has a value that differs from its last value by an increment or decrement, and identifier-4 and identifier-7 have the current values of identifier-5 or identifier-8.

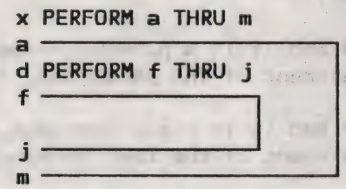
2. If an index name or identifier in the FROM phrase is changed while a PERFORM statement is executing, the change has no effect on the number of times the specified procedures are to be performed. However, changing the value of an index-name or an identifier in the VARYING phrase, or changing the value of an identifier in the BY phrase, does alter the number of times the procedures are executed.

3. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.

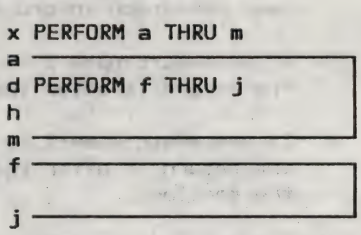
4. When only procedure-name-1 is specified, GO TO and PERFORM statements may occur within the procedure.

5. When a procedure within the range of a PERFORM statement includes another PERFORM statement, the range of the embedded PERFORM statement must be either totally contained within or totally removed from the range of the initial PERFORM statement:

Correct:

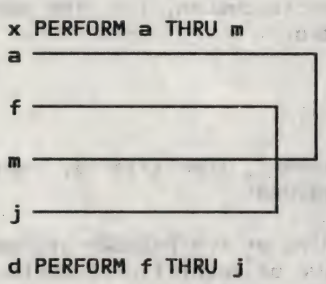


Correct:

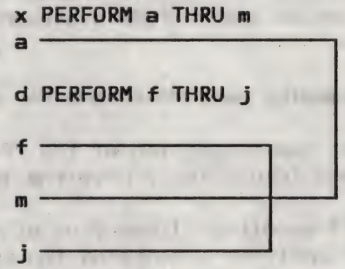


The range of procedures associated with a PERFORM statement may overlap or intersect the range associated with another PERFORM statement, if one PERFORM statement is not contained within the range of the other PERFORM statement. This is illustrated below:

Correct:



Incorrect:

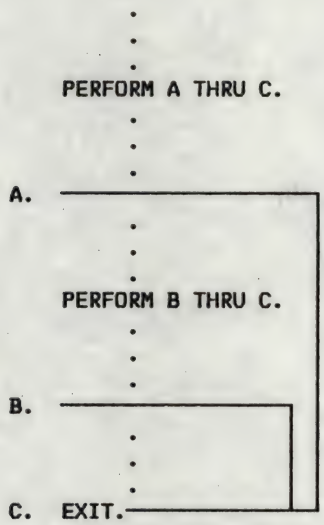


6. If the range of a PERFORM statement starts immediately after the PERFORM statement, the procedures within that range will be executed one extra time.

7. Nested PERFORM statements can have the same exit as the outer PERFORM statement,

   This exit must be the name of a paragraph consisting solely of the statement EXIT.

Example 2-96



For further information see the "COBOL User's Guide" [7], COBRUN Operand NESTPF.

8. The following information on segmentation is relevant here (for further details, refer to "Segmentation"):

   a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may, in its range, contain only the following procedures:

      - either sections all with a segment number less than 50, or

      - sections which, as a whole, are contained in one single segment whose segment number is greater than 49.

   b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may, in its range, contain only the following procedures:

      - either sections all with the same segment number as the section containing the PERFORM statement or

      - sections all with a segment number less than 50.

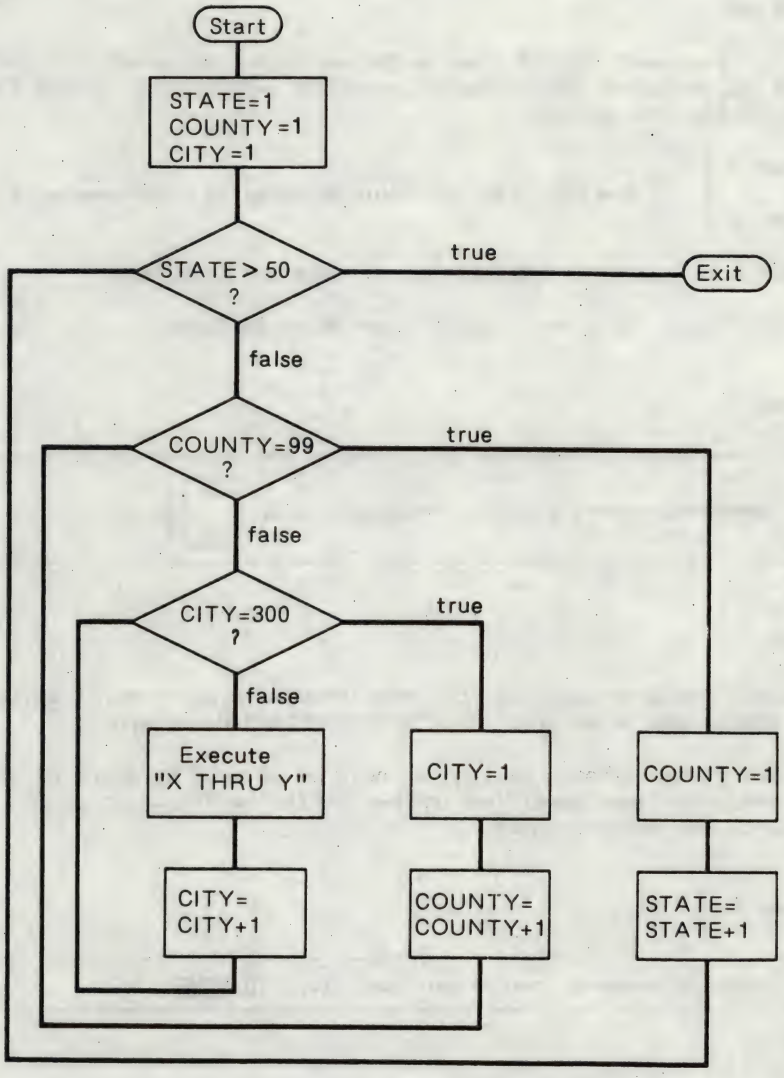
   c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 2-97 (format 4)

Statement:

```
        .
        .
        .
    PERFORM X THRU Y
        VARYING STATE FROM 1 BY 1
            UNTIL STATE IS GREATER THAN 50;
        AFTER COUNTY FROM 1 BY 1
            UNTIL COUNTY IS EQUAL TO 99;
        AFTER CITY FROM 1 BY 1
            UNTIL CITY IS EQUAL TO 300.
        .
        .
        .
```

Flowchart:


Start
STATE=1
COUNTY=1
CITY=1
STATE>50
?
true
Exit
false
COUNTY=99
?
true
false
CITY=300
?
true
false
Execute
"X THRU Y"
CITY=1
COUNTY=1
CITY=
CITY+1
COUNTY=
COUNTY+1
STATE=
STATE+1

SET STATEMENT

Function

The SET statement can be used to define reference points for table handling, to alter the status of external switches, or to set the value of condition variables.

Format 1
Format 2 } are required for table processing (see chapter 6).

Format 3 changes the status of external switches.

Format 4 sets the value of condition variables.

Format 3

```
SET {mnemonic-name-1 [mnemonic-name-2]... TO {ON  }}...
                                             {OFF }
```

Rules

1. Each mnemonic-name must be associated with an external switch whose status can be altered (see "SPECIAL-NAMES Paragraph").

2. The status of each switch can only be altered by means of the condition-name specified in the SPECIAL-NAMES paragraph and associated with the mnemonic-name.

Format 4

```
SET condition-name-1 [condition-name-2]... TO TRUE
```

Rule

condition-name-1 must be associated with a condition variable (see "Condition-Name-Condition").

Programming considerations

1. The literal specified in the VALUE clause and associated with condition-name-1 is entered in the condition variable according to the rules governing the VALUE clause (q.v.).

   If more than one literal is specified in the VALUE clause, the condition variable is set to the value of the first literal appearing in the VALUE clause.

2. If two or more condition-names are specified, they are treated as though a SET statement had been written for each individual condition-name.

Example 2-98

```
02  WORKDAY   PICTURE X.
    88 MONDAY VALUE "1".
    88 FRIDAY VALUE "5".
.
.
.
SET FRIDAY TO TRUE.
```

Once the SET statement has been performed, the data item WORKDAY (condition variable) will contain the literal assigned in the VALUE clause of FRIDAY (condition-name): "5".

Example 2-99

```
02  WEEKDAY    PICTURE X.
    88 WORKDAY  VALUE "1" "2" "3" "4" "5".
.
.
.
SET WORKDAY TO TRUE.
```

Once the SET statement has been performed, the data item WEEKDAY will contain the literal "1".

STOP STATEMENT

Function

The STOP statement halts execution of the object program either permanently or temporarily.

Format

```
STOP { RUN     }
     { literal }
```

Rules

1. The literal may be numeric, nonnumeric, or any figurative constant except ALL literal.
2. If the literal is numeric, it must be an unsigned integer.
3. If a STOP statement with RUN phrase appears in a sentence, then it must be the only statement in that sentence, or it must be the last statement in a sequence of imperative statements.
4. The STOP statement with the RUN phrase specified terminates execution of the object program, and returns control to the operating system.
5. If STOP literal is used, the literal is communicated to the system operator. In this case, only the system operator can resume the object program. Continuation of the object program begins with the next executable statement.

Programming considerations

1. If the number of characters of the nonnumeric literal exceeds the hardware capability of the master console or subconsole, then more than one physical output operation will be performed to output the literal.
2. All files should be closed before a STOP RUN statement is issued.

Example 2-100

```
STOP "PLEASE MOUNT NEXT REEL; THEN RESTART.".
```

When this statement is executed, execution of the object program stops; and the message PLEASE MOUNT NEXT REEL; THEN RESTART. appears on the operator's console. When the operator has mounted the tape, he restarts the program.

For further information see the "COB1 User's Guide" [7].

2

STRING STATEMENT

Function

The STRING statement moves and juxtaposes the partial or complete contents of two or more data items into a single data item.

Format

```
STRING {identifier-1} [{identifier-2}] ... DELIMITED BY {identifier-3}
       {literal-1   }  {literal-2   }                    {literal-3   }
                                                         {SIZE        }

[[{identifier-4} [{identifier-5}] ... DELIMITED BY {identifier-6}]]...
  {literal-4   }  {literal-5   }                   {literal-6   }
                                                   {SIZE        }

INTO identifier-7 [WITH POINTER identifier-8]
[ON OVERFLOW imperative statement]
```

Rules

1. Each literal may be any figurative constant, except ALL literal.

2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-8, must be described implicitly or explicitly as USAGE IS DISPLAY.

3. Where identifier-1, identifier-2, ..., identifier-3 are elementary numeric data item, they must be described as integers without the symbol P in their PICTURE strings.

All references below to identifier-1, identifier-2, identifier-3, literal-1, literal-2, literal-3 apply equally to identifier-4, identifier-5, identifier-6, literal-4, literal-5, literal-6 and for all recursions thereof.

4. identifier-1, literal-1, identifier-2, literal-2 represent the sending items. Identifier-7 represents the receiving item.

5. identifier-3, literal-3 represent delimiters, i.d. they mark a character string up to which the contents of a sending item should be moved. If the SIZE phrase is used, the contents of the complete data item defined by identifier-1, literal-1, identifier-2, literal-2 are moved. When a figurative constant is used as the delimiter, it is a single character nonnumeric literal.

6. When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit one character data item whose usage is DISPLAY.

7. identifier-7 must not represent an edited data item and must not be described with the JUSTIFIED clause.

8. identifier-8 is a counter item and must be described as an elementary numeric integer data item of sufficient size to accommodate the size of the data item referenced by identifier-7 plus the value 1. The symbol P is prohibited in the picture-string of identifier-8.

Programming considerations

1. When the STRING statement is executed, the transfer of data is governed by the following rules:

   a) The sending items literal-1, literal-2 or the contents of identifier-1, identifier-2 are transferred to the receiving item referenced by identifier-7 in accordance with the rules for alphanumeric-to-alphanumeric moves, except that no space filling will be provided.

   b) If the DELIMITED BY phrase is specified without the SIZE phrase, the contents of identifier-1, identifier-2, or the value of literal-1, literal-2, is transferred character-by-character to the receiving data item, beginning with the leftmost character and continuing from left to right until the end of the sending data item is reached or until the character(s) specified by the delimiter referenced by literal-3, or the contents of identifier-3, are encountered.
   The delimiter is not transferred.

   c) If the DELIMITED BY phrase is specified with the SIZE phrase, the entire contents of the sending items referenced by literal-1, literal-2, or identifier-1, identifier-2, are transferred to the receiving item referenced by identifier-7 until all data has been transferred or the end of identifier-7 has been reached. The transfer takes place in the sequence specified in the STRING statement.

2. If the POINTER phrase is specified, the counter item referenced by identifier-8 is explicitly available to the user, i.e. an initial value greater than 0, and of adequate size to contain a value of the length of identifier-7 plus one byte, must be set to the user.

3. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-8 referencing a data item with an initial value of 1.

4. When moved to the receiving item identifier-7, each character is transferred separately from the sending field to the character position of identifier-7 which is determined by the value of the counter item identifier-8. Each time a character is moved, identifier-8 is incremented by 1. This is the only manner in which the value of identifier-8 changes during execution of the STRING statement.

5. At the end of execution of the STRING statement, only the portion of the receiving item referenced by statement-7 into which characters were moved is changed. The rest of identifier-7 remains the same.

6. If at any time during or after the initialization of the STRING statement, but before processing is completed, the value associated with the counter item referenced by identifier-8 is either less than one or exceeds the number of character positions in the receiving item referenced by identifier-7, no (further) data is transferred to identifier-7, and the imperative statement in the ON OVERFLOW phrase, if specified, is executed.

7. If the ON OVERFLOW phrase is not specified when one of the conditions described above is encountered, control is transferred to the end of the STRING statement.

Example 2-101

```
ID DIVISION.
PROGRAM-ID. STRNG.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  FIELD1   PIC X(16)  VALUE "INITIAL RESTRICTION".
77  FIELD2   PIC X(12)  VALUE "VALUE RANGE".
77  FIELD3   PIC X(25)  VALUE SPACES.
77  FIELD4   PIC 99     VALUE 3.
PROCEDURE DIVISION.
EXAMPLE.
    DISPLAY "BEFORE STRING".
    PERFORM DISPLAY-FIELDS.
    STRING FIELD1, FIELD2 DELIMITED BY "R",
           "INVALID" DELIMITED BY SIZE
      INTO FIELD3 WITH POINTER FIELD4
        ON OVERFLOW
        DISPLAY "ERROR".
    DISPLAY "AFTER STRING".
    PERFORM DISPLAY-FIELDS.
    STOP RUN.
DISPLAY-FIELDS.
    DISPLAY "FIELD1 = *" FIELD1 "*".
    DISPLAY "FIELD2 = *" FIELD2 "*".
    DISPLAY "FIELD3 = *" FIELD3 "*".
    DISPLAY "FIELD4 = *" FIELD4 "*".
```

Result:

FIELD1 before and after STRING (identifier-1)	I N I T I A L - R E S T R I C T I O N
FIELD2 before and after STRING (identifier-2)	V A L U E - R A N G E
FIELD3 before STRING	- -
FIELD3 after STRING (identifier-7)	- - I N I T I A L - V A L U E - I N V A L I D - -
FIELD4 before STRING	0 3
FIELD4 after STRING (identifier-8)	2 2

SUBTRACT STATEMENT

Function

The SUBTRACT statement is used to subtract the value of a numeric item, or the sum of two or more values of numeric data items, from one or more items.

Format 1 of the SUBTRACT statement stores the difference in one of the operands. In a SUBTRACT statement, more than one subtraction may be specified by supplying more than one result item.

Format 2 of the SUBTRACT statement uses the GIVING phrase.

Format 3 of the SUBTRACT statement uses the CORRESPONDING phrase.

Format 1

```
SUBTRACT {identifier-1} [identifier-2]
         {literal-1   } [literal-2   ]...

         FROM identifier-m [ROUNDED] [identifier-n [ROUNDED]]...

        [ON SIZE ERROR imperative-statement]
```

Rules for format 1

1. Each identifier must refer to an elementary numeric data item.

2. The composite of operands determined by using all of the operands in a given statement, with the exception of the data item following the word GIVING, must not exceed 18 digits (see "Arithmetic Statements").

3. All literals and identifiers preceding the word FROM are added together, and the sum is subtracted from the current values of identifier-m, identifier-n ... . The results of the subtraction are stored as the new values of identifier-m, identifier-n ... .

Example 2-102 (format 1)

Statement	PICTURE of result field	Calculation
SUBTRACT A, B FROM D	999	D - (A + B) stored in in D as nnn
SUBTRACT A FROM B,C	9(3) for B 9(5) for C	B - A stored in B as nnn C - A stored in C as nnnnn

Format 2

```
SUBTRACT {identifier-1} [identifier-2] ...
         {literal-1   }  [literal-2   ]

    FROM {identifier-m}
         {literal-m   }

    GIVING identifier-n [ROUNDED] [identifier-o [ROUNDED]]...

    [ON SIZE ERROR imperative-statement]
```

Rules for format 2

1. Each identifier preceding the word GIVING must refer to an elementary numeric item.
2. identifier-n, identifier-o, may refer both to an elementary numeric item and to an elementary numeric-edited data item.
3. The composite of operands determined by using all of the operands in a given statement, except for the data items which follow the word GIVING, must not be greater than 18 digits (see "Arithmetic Statements").
4. All literals or identifiers preceding the word FROM are added together, and the sum is subtracted from literal-m or identifier-m. The result of the subtraction is the new value of identifier-n, identifier-o, ...

Example 2-103 (format 2)

Statement	PICTURE of result field (C))	Calculation
SUBTRACT A, B FROM 100 GIVING C.	9(5)	100 - (A + B) stored in C as nnnnn

Format 3

```
SUBTRACT {CORR          } identifier-1
         {CORRESPONDING }

         FROM identifier-2 [ROUNDED]

        [ON SIZE ERROR imperative-statement]
```

Rules for format 3

1. Each identifier must refer to a group item.

2. The composite of operands, which is determined separately for each pair of corresponding data items, must not be greater than 18 digits (see "Arithmetic Statements").

3. When the CORRESPONDING phrase is used, the elementary items within the first operand (identifier-1) are subtracted from the corresponding elementary items within the second operand (identifier-2). The results are stored in the items of the second operand.

For further rules, see under "Options in Arithmetic Statements"; the CORRESPONDING, ROUNDED and SIZE ERROR phrases are all described in this chapter.

Example 2-104 (format 3)

Refer to the description of the CORRESPONDING phrase for an example of the use of this phrase.

UNSTRING STATEMENT

Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

```
UNSTRING identifier-1

    [DELIMITED BY [ALL] {identifier-2} [OR [ALL] {identifier-3}]...]
                        {literal-1   }          {literal-2   }

    INTO identifier-4 [DELIMITER IN identifier-5] [COUNT IN identifier-6]

        [identifier-7 [DELIMITER IN identifier-8] [COUNT IN identifier-9]]...

    [WITH POINTER identifier-10] [TALLYING IN identifier-11]

    [ON OVERFLOW imperative-statement]
```

Rules

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant, except ALL literal.

2. identifier-1, identifier-2, identifier-3, identifier-5, identifier-8 must reference data items described, implicitly or explicitly, as alphanumeric.

3. identifier-4, identifier-7 may be described as either

   alphabetic (except that the symbol B may not be used in the PICTURE string), or
   alphanumeric or numeric (except that the symbol P may not be used in the PICTURE string)

   and must be described, implicitly or explicitly, as USAGE IS DISPLAY.

4. identifier-6, identifier-9, identifier-10, identifier-11 must reference integer data items (except that the symbol P may not be used in the PICTURE string).

5. All references to identifier-2, literal-1, identifier-4, identifier-5, identifier-6 apply equally to identifier-3, literal-2, identifier-7, identifier-8, identifier-9, and all recursions thereof.

6. identifier-1 represents the sending area.

7. identifier-4 represents the receiving area, identifier-5 the receiving area for delimiters.

8. literal-1 and identifier-2 are delimiters.

9. identifier-6 represents the count of the number of characters encountered within identifier-1 up to the delimiter, i.e. the number of characters within identifier-1 which are to be moved to identifier-4. This value does not include a count of the delimiter character(s).

10. The data item referenced by identifier-10 contains a value that indicates a character position within the area referenced by identifier-1 (relative to the beginning of that area).

11. The data item referenced by identifier-11 is a counter for the number of receiving fields addressed in an UNSTRING statement.

12. literal-1 or identifier-2 may contain any character from the native character set.

13. DELIMITER IN and COUNT IN can be used only in conjunction with DELIMITED BY.

Programming considerations

1. No identifier may be defined with the level number 88.

2. When a figurative constant occurs as a delimiter, it represents a single-character nonnumeric literal.

3. When the ALL phrase is specified, contiguous occurrences of literal-1 or identifier-2 are treated as if they were only one occurrence, and one occurrence of literal-1 or identifier-2 is moved to the data item referenced by identifier-5.

4. When two contiguous delimiters are encountered, the current receiving area is space filled if it is described as alphabetic or alphanumeric, or zero filled if it is described as numeric.

5. literal-1, identifier-2 represent delimiters. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given, to be recognized as a delimiter.

6. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is/are considered to be a single delimiter. No character(s) in the sending field can be considered as part of more than one delimiter. Delimiters cannot overlap.

   Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

7. Once the UNSTRING statement is initialized, identifier-4 represents the current receiving area. Data is transferred from identifier-1 to identifier-4 according to the following rules:

   a) If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by identifier-10.

      If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position of identifier-1.

b) If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by literal-1 or the value of the data item referenced by identifier-2 is encountered.

   If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area.

   However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

   If the end of identifier-1 is reached before a delimiter is encountered, the examination terminates with the character examined last.

c) The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary data item, and are moved into the current receiving area according to the rules for the MOVE statement (see "MOVE Statement").

d) If the DELIMITER IN phrase is specified, the delimiting character(s) are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement (see "MOVE Statement").

e) If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into the area referenced by identifier-6 according to the rules for an elementary move.

f) If the DELIMITED BY phrase is specified, the string of characters referenced by identifier-1 is further examined beginning with the first character to the right of the delimiter.

   If the DELIMITED BY phrase is not specified, the examination will continue from the character immediately following the last character to be moved.

g) After the data is transferred to identifier-4, the current receiving area is represented by identifier-7.

   The behavior described above is repeated until either all the characters in identifier-1 are exhausted or until there are no more receiving areas.

8. The initialization of the data items associated with the POINTER or TALLYING phrase is the responsibility of the user.

9. The contents of identifier-10 are incremented by 1 each time a character is checked in the data item indicated by identifier-1.

   When the execution of an UNSTRING statement with a POINTER phrase is completed, identifier-10 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

10. When the execution of an UNSTRING statement with a TALLYING phrase is completed, identifier-11 contains a value equal to its initial value plus the number of receiving data items addressed.

11. Either of the following situations causes an overflow condition:

    a) If an UNSTRING is initiated, and the value in the data item referenced by identifier-10 is less than 1 or greater than the size of the data item referenced by identifier-1.

    b) If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

12. When an OVERFLOW condition exists, the UNSTRING operation is terminated.

    If an ON OVERFLOW phrase has been specified, the imperative statement given in this phrase is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next statement to be executed.

13. Subscripts and indexes for the identifiers are analyzed as follows:

    a) If the items identifier-1, identifier-10, identifier-11 are subscripted or indexed, the index value for these items is calculated once only, namely, just before the UNSTRING statement is executed.

    b) Any subscripting or indexing associated with identifier-2, identifier-3, identifier-4, identifier-5, identifier-6 is evaluated immediately before the data are transferred to the respective data item.

Example 2-105

```
ID DIVISION.
PROGRAM-ID. UNSTRNG.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
 .
 .
 .
01  FIELD     PIC  X(12)  VALUE  "ABCDEFGHIJKL".
01  AREA1.
    02  PART1 PIC X       VALUE  SPACES.
    02  PART2 PIC XX      VALUE  SPACES.
    02  PART3 PIC XXX     VALUE  SPACES.
01  NUMBER1   PIC 99      VALUE  ZERO.
PROCEDURE DIVISION.
BEG.
    DISPLAY "BEFORE UNSTRING".
    PERFORM DISPLAY-FIELDS.
*
    UNSTRING FIELD
        DELIMITED BY "E"  OR  "H"  OR  "K"  OR "L"
        INTO PART3,  PART2,  PART1
        TALLYING IN NUMBER1.
*
    DISPLAY "AFTER UNSTRING".
    PERFORM DISPLAY-FIELDS.
    STOP RUN.
DISPLAY-FIELDS.
    DISPLAY  "FIELD  = *"  FIELD  "*".
    DISPLAY  "PART1  = *"  PART1  "*".
    DISPLAY  "PART2  = *"  PART2  "*".
    DISPLAY  "PART3  = *"  PART3  "*".
    DISPLAY  "NUMBER = *"  NUMBER "*".
```

Result:

BEFORE UNSTRING	AFTER UNSTRING
FIELD = *ABCDEFGHIJKL*	FIELD = *ABCDEFGHIJKL*
PART1 = * *	PART1 = *I*
PART2 = * *	PART2 = *FG*
PART3 = * *	PART3 = *ABC*
NUMBER = *00*	NUMBER = *03*

3 SEQUENTIAL FILE ORGANIZATION

3.1 FILE CONCEPTS

A file is a collection of data records that can be moved to, or read from, a volume. The user defines the organization of the file as well as the mode and order of file processing.

The organization of a file describes its logical structure. There are sequential, indexed, relative, and direct types of file organization. The file organization which is defined at the time the file is constructed cannot be changed later on. This same rule applies to the maximum record size of a particular file, as defined in the file and record description entries.

3.1.1 Sequential Organization

When sequential file organization is used, the logical data records are placed on the file or read sequentially, i.e. in the order in which they were generated (or in reverse order if the REVERSED phrase was specified in the OPEN statement for a magnetic-tape unit). This file organization is referred to as "sequential organization".

This type of file organization must be used for magnetic-tape or unit-record files and may be used for disk-storage files. Sequentially organized files require no key for record processing.

3.1.2 Sequential Access to Data Records of a Sequential File

A sequential file can only be processed sequentially, i.e. the records are either read or written in the order predefined by the file. Data records may thus be created and read; with sequential files on disk-storage devices, data records may also be updated. The only prerequisite is a READ statement followed immediately by a REWRITE.

3.1.3 I-O Status

The I-O status is a value which is transferred to a two-character data item for the purpose of indicating to the COBOL program the status of an input/output operation. This takes place during execution of a CLOSE, OPEN, READ, REWRITE or WRITE statement and prior to execution of any associated imperative statement and any corresponding USE AFTER STANDARD EXCEPTION procedure. The I-O status value is transferred only if the FILE STATUS clause is specified in the FILE-CONTROL paragraph (see FILE STATUS clause).

At the termination of an I-O operation the values of the I-O status have the following meaning:

1. Execution successful
   The I-O statement was processed normally.

2. Execution unsuccessful: AT END condition
   A READ statement terminated abnormally due to an AT END condition.

3. Execution unsuccessful: unrecoverable error
   The I-O statement terminated abnormally due to the occurrence of an unrecoverable error precluding further processing of the file. An unrecoverable error cannot be rectified within the COBOL program.

4. Execution unsuccessful: logical error
   An I-O statement terminated abnormally due to an illegal sequence of I-O operations for this file.

5. Conflict between the permanent characteristics of a file and those described in the program
   The I-O statement terminated abnormally because the permanent file characteristics are incompatible with those given in the COBOL program.

6. System error
   The I-O statement terminated abnormally due to a system error.

The table below shows the values of the I-O status and their meanings:

I-O status	Meaning
	Execution successful
00	- The I-O statement terminated normally. No further information regarding the I-O operation is available.
04	- Record length conflict: A READ statement terminated normally. However, the length of the record read lies outside the limits defined in the record description entry for this file.
05	- AN OPEN statement terminated normally. However, the referenced file indicated by the OPTIONAL entry in the SELECT clause was nonexistent at the time the OPEN statement was executed.
	Execution unsuccessful: AT END condition
10	- An attempt was made to execute a READ statement. However, the next logical record was nonexistent as the end-of-file was encountered.
15	- A first attempt was made to execute a READ statement for a nonexistent file with OPTIONAL entry.
16	- An attempt was made to execute a READ statement although an AT END condition had already been encountered.
	Execution unsuccessful: unrecoverable error
30	- No further information regarding the I-O operation is available.
34	- An attempt was made to write outside the sequential file limits set by the system.
35	- An attempt was made to execute an OPEN statement with INPUT phrase specified for a nonexistent file with no OPTIONAL entry specified.
38	- An attempt was made to execute an OPEN statement for a file previously locked by the LOCK phrase.
39	- The OPEN statement was unsuccessful because a) one or more of the operands FCBTYPE, RECFORM or RECSIZE in the FILE command were supplied with values that conflict with the corresponding implicit or explicit program specifications, or b) record length errors occurred for input files (catalog check), or c) the record length is greater than the BLKSIZE in the FILE command, or d) a RECSIZE ≠ 0 was specified in the FILE command for an output file with RECFORM = V.

I-O status	Meaning
	Execution unsuccessful: logical error
41	- An attempt was made to execute an OPEN statement for a file which was already open.
42	- An attempt was made to execute a CLOSE statement for a file which was not open.
43	- While accessing a disk file opened with OPEN I-O: The most recent I-O statement executed prior to a REWRITE statement was not a successfully executed READ statement.
44	- Exceeding the area limits: a) An attempt was made to execute a WRITE or REWRITE statement. However, the length of the record exceeds the area allowed for this file. b) An attempt was made to execute a REWRITE statement. However, the record to be rewritten did not have the same length as the record to be replaced.
46	- An attempt was made to execute a READ statement for a file in INPUT or I-O mode. However, there is no next valid record since the preceding READ statement terminated abnormally without causing an AT END condition.
47	- An attempt was made to execute a READ statement for a file not in INPUT or I-O mode.
48	- An attempt was made to execute a WRITE statement for a file not in OUTPUT, I-O or EXTEND mode.
49	- An attempt was made to execute a REWRITE statement for a file not in I-O mode.
	Other unsuccessful executions
90	- System error; nor further information available regarding the cause.
91	- System error; a system call terminated abnormally, e.g. password error.

3.2 LANGUAGE ELEMENTS OF THE ENVIRONMENT DIVISION

3.2.1 INPUT-OUTPUT SECTION

Function

The INPUT-OUTPUT SECTION is used to define each file, to name its external devices, to assign each file to one or more input/output devices, and also to provide the information needed for an efficient transmission of data between the input/output devices and the object program.

This section is divided into two paragraphs:

the FILE-CONTROL paragraph, which names the files used in the program and assigns each file to external devices, and

the I-O-CONTROL paragraph, which indicates special input/output techniques.

Format



margin indication

```
INPUT-OUTPUT SECTION.

[FILE-CONTROL. [file-control-entry.     ]...]

[I-O-CONTROL.  [input-output-control-entry.]]
```

Programming considerations

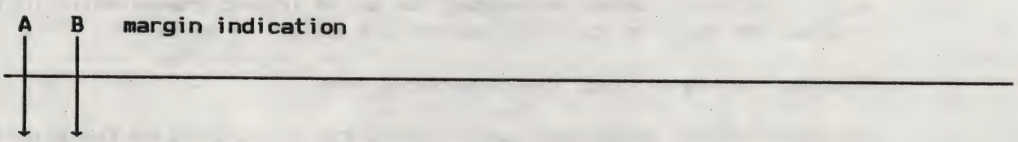
1. All sections and paragraphs must begin at margin A.
2. The entire INPUT-OUTPUT SECTION is optional.

3.2.1.1 FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph is used to give each file a name. The files are assigned to one or more input/output devices, and the information required for file processing is made available. This information indicates how the data is organized and how to access it.

Format




```
FILE-CONTROL

    SELECT clause
    ASSIGN clause
   [ORGANIZATION clause]

   [ACCESS MODE clause]

   [RESERVE clause]

   [FILE STATUS clause]
```

Rules

1. The heading of the FILE-CONTROL paragraph must start at margin A. All subsequent entries must start at margin B.

2. The SELECT clause must be the first entry in the FILE-CONTROL paragraph. It is followed immediately by the ASSIGN clause. All other clauses may be specified in any order.

In the discussion below, the SELECT and ASSIGN clauses are described first, followed by the remaining clauses in alphabetical order.

SELECT CLAUSE

Function

The SELECT clause is used to name each file in a program.

Format 1 is used to name all files except sort-files.

Format 2 see chapter "Sort-Merge Feature".

Format 1

SELECT [OPTIONAL] file-name

Rules

1. file-name stands for the name with which a file is referenced in the source program (internal file-name). Each file-name used in a program may only occur once in the SELECT clause.

2. Each file specified in a SELECT clause must have a file description (FD) entry in the Data Division of the source program.

3. The OPTIONAL clause is required for input files that are not necessarily present at object time. When a file is not present at object time, the first READ statement for that file passes control to its associated AT END condition.
   The OPTIONAL clause applies only to files opened as OPEN INPUT.

4. Up to 254 files may be specified in the SELECT clauses within a source program.

5. The first 8 characters of file-name are used as the file link name, provided the COBRUN operand LINK was not specified when the program was compiled (for further information on this point see the ASSIGN clause and the "COB1 User's Guide" [7]: COBRUN Operands).
   Since the link name cannot have more than 8 characters, file-name must be unique in these positions. If file-name is longer, is will be truncated for the link name. If it then ends with a hyphen in position 8, a warning will be issued during compilation and the hyphen will be converted into a number sign (#).

6. If no FILE command was issued for a file at program runtime, file-name is used is a BS2000 file name in a FILE macro. In this case, all characters in file-name (up to 30) will be used.

ASSIGN CLAUSE

Function

The ASSIGN clause assigns an external device to a file of the COBOL program. One ASSIGN clause is required for each file in the program.

Format 1 is used for all files except sort-files.

Format 2 See chapter "Sort-Merge Feature".

Format 1

```
ASSIGN TO [integer] implementor-name-1 [implementor-name-2]...
```

Rules

1. The implementor-name designates a device and must be:

   either

   a) a system-name chosen from the following list:

SYSIPT or SYSIN	System logical input device. Both use the symbolic device name SYSIPT.
SYSOUT or SYSLST	System logical output device. Both use the symbolic device name SYSLST.
SYSLST01-SYSLST99	System logical output files SYSLST01-SYSLST99.
SYSPUNCH or SYSOPT	System logical output device (80-column card format). Both use the symbolic device name SYSOPT.

   or

   b) `class-[device-][organization-]name`

   or

   c) `[[class-]device-[organization-]]name-1`

   where c) can only be specified in conjunction with COBRUN LINK (see BS2000 "COB1 User's Guide").

   Class is a two-character field (optional in c only) that identifies a device class.

   UR specifies a device for sequential access (printer, reader, punch)

UT specifies a device for sequential access (tape, disk-storage device)

DA specifies a device for direct access (disk-storage device).

Device is an optional field that contains a symbolic device name specifying the type of device to which the file is to be assigned.

Class	Device name	Device assigned
UR	PRINTER 1) PRINTERDOD 1) PRINTERANS 1) READER	any single-carriage printer any single-carriage printer any single-carriage printer any card reader
UT	TAPE TAPISO TAPDISC	any tape, EBCDIC code1) any tape, ISO-7-bit code1) any tape, interchangeable with disk-storage device
UT, DA	DISC 2)	disk-storage device

Table 3-1 Device names

1) See "WRITE Statement".

2) The DISC entry is used only for checking the validity of block/record length at compile time and does not affect the actual allocation of devices at object time. At object time, it is the user's responsibility to provide correct device assignments.

Organization is a single-character field specifying the file organization. For sequential files, S is the only character permitted.

Name is a six-character field that gives the symbolic name of the device to which the file is assigned. It must have the following format:

SYSnnn (where nnn is an integer between 000 and 244)

SYSnnn is not evaluated. The required LINK name is taken from the first 8 characters of the file-name given in the SELECT clause.

Name-1 in c) stands for the required link-name (see the COBRUN LINK description in the BS2000 COB1 User's Guide).

2. Integer indicates the number of devices assigned to the file. It is present only for reasons of compatibility, and is treated by the compiler as comments.

Programming consideration

File organization may also be specified in the ORGANIZATION clause (q.v.).

ACCESS MODE CLAUSE

Function

The ACCESS MODE clause determines the manner in which the records of a file are to be accessed.

Format

```
ACCESS MODE IS SEQUENTIAL
```

Rules

1. SEQUENTIAL means that data records are read or written sequentially, i.e. the next logical data record of the file is made available when executing a READ statement, or the next logical data record is placed on that file when executing a WRITE statement.

2. If the ACCESS MODE clause is not supplied, ACCESS IS SEQUENTIAL is assumed.

FILE STATUS CLAUSE

Function

The FILE STATUS clause specifies a data item whose contents are used for interrogating the status of input/output operations during processing. In addition, by specifying a further data item, the DMS code is made available.

Format

```
FILE STATUS IS data-name-1 [, data-name-2]
```

Rules

1. data-name-1 and data-name-2 are only allowed one qualifier.

2. data-name-1 and data-name-2 must be defined in the LINKAGE SECTION or WORKING-STORAGE-SECTION of the Data Division. Data-name-1 must be a two-byte numeric (DISPLAY only) or alphanumeric item. data-name-2 must be a 6-byte group item in the form

```
Om      data-name-2.
Om+1    data-name-x PIC 9(3) COMP.
Om+1    data-name-y PIC X(4).
```

3. If the FILE STATUS clause is specified, the runtime system will

   a) move the I-O status to data-name-1, and

   b) set data-name-2, if specified, as follows depending on the values given in data-name-1 and the DMS code:

      b1) If data-name-1 has the value 0, then data-name-2 is not given a new setting, i.e. its contents are considered undefined.

      b2) If data-name-2 has a non-zero value and the DMS code is 0, the left portion (first two bytes) of data-name-2 is set with 0 while the right portion (remaining 4 bytes) remains undefined. If, however, the DMS code is non-zero, the left portion of data-name-2 is given the value 128 and the right portion the DMS code (for further information see the DMS manuals).

   This takes place during execution of each OPEN, CLOSE, READ, WRITE or REWRITE statement which references that file, and before execution of each corresponding USE procedure (see "I-O Status").

ORGANIZATION CLAUSE

Function

The ORGANIZATION clause defines the logical structure of a file.

Format

```
[[ORGANIZATION IS] SEQUENTIAL]
```

Rules

1. SEQUENTIAL is the default value supplied by the compiler.

2. The file organization can also be specified in the ASSIGN clause using the organization item of the implementor-name (see "ASSIGN Clause").

   The entry can also be made in both places, in which case it must be identical. If omitted, ORGANIZATION IS SEQUENTIAL is assumed.

Programming consideration

File organization is defined at file construction time and is not, therefore, subject to modification.

RESERVE CLAUSE

Function

The RESERVE clause allows the user to modify the number of input/output areas (buffers) the compiled program will be allocated by the compiler.

Format

```
RESERVE [integer] [AREA  ]
                  [AREAS ]
```

Rules

1. integer must be an unsigned positive integer.

2. RESERVE AREA
   One input/output area is reserved.

3. RESERVE integer AREA
   The number of input/output areas indicated by integer are reserved.

4. RESERVE integer AREA
   The number of input/output areas indicated by integer is also reserved in addition to the input/output area which is generated by default.

Programming considerations

1. RESERVE integer AREA
   If the COBRUN operand ANSICOB is specified in the program, the RESERVE clause is interpreted according to the processing mode used previously in ANSICOB, i.e. the number of input/output areas indicated by integer is added to the input/output area which is generated as a default value.

2. No more than two input/output areas may be created.

3. Only one input/output area may be specified for a file which was opened in I/O mode.

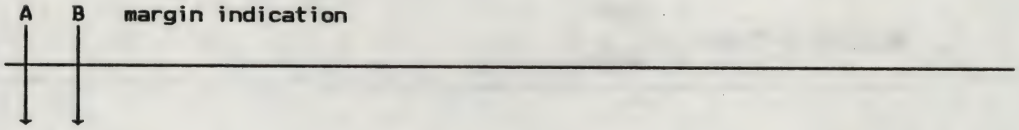
4. Only one input/output area may be specified for a file assigned to a system device (SYSIN, SYSIPT, SYSLST, SYSOUT, SYSPUNCH, SYSOPT).

3.2.1.2 I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph defines the events at whose occurrence restart points are to be established, and it specifies the memory area which is to be shared by the various files. Also, it indicates the location of files on multiple-file reels, and defines special input-output conditions.

Format




```
I-O-CONTROL.

    [RERUN clause]...

    [SAME AREA clause]...

    [MULTIPLE FILE TAPE clause]... .
```

Rule

I-O-CONTROL must be written starting at margin A. All subsequent entries must be written in Area B.

For better readability, the clauses are described in the following subsections in alphabetical order.

MULTIPLE FILE TAPE CLAUSE

Function

The MULTIPLE FILE TAPE clause is required when more than one file shares the same physical reel of tape.

Format

```
MULTIPLE FILE TAPE CONTAINS file-name-1 [POSITION integer-1]
        [file-name-2 [POSITION integer-2]]...
```

Rule

integer can have any value between 1 and 3315.

Programming considerations

1. When all file-names are supplied in the same order in which they appear on a reel, the POSITION phrase may be omitted.
2. If any of the files is not specified, then the positions must be supplied as relative to the beginning of the tape.
3. Irrespective of the number of files that share the reel of tape, only those used by the compiled object program need be specified.
4. Only one file may be opened on the same reel at any given time.
5. REWIND can be carried out when the last file of the tape has been processed.

For further information see the "COB1 User's Guide" [7].

RERUN CLAUSE

Function

A RERUN clause indicates where and when restart-point records are to be issued. A restart-point record describes the status of an object program at a specified point during program execution. It is produced automatically by the operating system upon the request of the object program and contains all information necessary to restart the program from that point. The RERUN clause controls such requests by the COBOL object program.

Format

```
RERUN [ON {file-name-1      }]
      [   {implementor-name}]

                                  [[END OF] {REEL}]
                                  [         {UNIT}]
                EVERY {                             } OF file-name-2
                      {integer-1 RECORDS            }
                      {integer-2 CLOCK UNITS        }
                      {condition-name               }
```

Rules

1. file-name-1 must be the name of a sequential output file; it must be described in the FILE section.

2. implementor-name has the following format:
   SYSnnn.
   where 000 ≤ nnn ≤ 244.
   nnn determines the processing mode. If nnn ≤ 200, restart points are written alternately to two restart files. This enables the restart to be made from the two most recent restart points.

3. If the same implementor-name is also used in the SELECT clause, it must be associated with a tape file.

4. The END OF REEL/UNIT phrase may be used only when file-name-2 describes a sequential file. REEL and UNIT are synonymous.

5. implementor-name must be specified in the RERUN clause when integer-1 RECORDS or integer-2 CLOCK-UNITS is used.

6. Subject to the following restrictions, more than one RERUN clause may be specified for the same file-name-2.

   a) When specifying more than one integer-1 RECORDS phrase, the same file-name-2 must appear only once in them.

   b) When specifying more than one END OF REEL or END OF UNIT phrase, the same file-name-2 must appear only once in them.

7. file-name-1 or implementor-name specifies the file to which the restart points are to be output.

8. Restart points are written as follows:

   a) If file-name-1 is specified, the restart points are output to each reel or volume assigned to file-name-1.

   b) If implementor-name is specified, the restart points are written as follows:
   When implementor-name is assigned to a file-name from the SELECT clause, the file involved must be a tape file. The effect on restart point generation is the same as if the file-name had been specified directly in the RERUN clause (see 8a).

   When implementor-name is not assigned to a file from a SELECT clause, a disk storage file must be assigned at runtime. Only in this case will restart points be written to disk storage.

9. There are four types of RERUN clauses, depending on the conditions under which restart points are requested.

   a) END OF REEL or END OF UNIT without specification of the ON clause: the restart points are written to file-name-2, which must specify an output file.

   b) END OF REEL or END OF UNIT with file-name-1 specified in the ON clause: the restart points are written to file-name-1, which must refer to an output file. Additionally, the usual end-of-reel handling is carried out for file-name-2. file-name-2 may be either an input or output file.

   c) END OF REEL or END OF UNIT with implementor-name in the ON clause:
   The restart points are written to a separate file (see 8b). File-name-2 may be an input or an output file.

   d) integer-1 RECORDS:
   The restart points are written to the file specified by file-name-1 or implementor-name, respectively, whenever integer-1 records have been processed. File-name-2 may be either an input or an output file with any organization or type of access; it must not be referenced in the USING/GIVING phrase or in an INPUT/OUTPUT procedure during sort (see "Sort/Merge" chapter).

10. The CLOCK-UNITS phrase is treated by the compiler as comments.

11. The condition-name-entry, too, is treated by the compiler as comments.

SAME AREA CLAUSE

Function

The SAME AREA clause indicates that two or more files are to share a specified input/output area during program execution.

Format 1 of the SAME AREA clause specified all files except sort-files unless RECORD is supplied.

Format 2 see chapter "Sort-Merge Feature".

Format 1

```
SAME [RECORD] AREA FOR file-name-1 {file-name-2}...
```

Rules

1. More than one SAME AREA clause may be included in a program. In this case, the following must be observed:

   A specific file-name must not appear in more than one SAME AREA clause. The same is true of the SAME RECORD AREA clause.

   A specific file-name may concurrently appear in a SAME RECORD AREA clause. In this case, all file-names appearing in the SAME AREA clause must also appear in the SAME RECORD AREA clause. The SAME RECORD AREA clause may also contain other file-names that do not appear in the SAME AREA clause.

2. The SAME AREA clause indicates that the specified files (no sort-files) are to share the input/output areas assigned to them.

3. The SAME RECORD AREA clause indicates that the specified files are to share the same storage for processing the current logical record.

   A logical record in the SAME RECORD AREA clause is considered a logical data record of all files opened for OUTPUT whose names are supplied in that SAME RECORD AREA clause. It is also a logical data record of the file (in this clause) from which the most recent input occurred. This is equivalent to an implicit overwriting of all data record areas, where the data records are aligned on the leftmost character position.

Programming considerations

1. If SAME AREA is used, only one file may be open at any given time.

2. If the RECORD phrase is used, all specified files may be open at the same time.

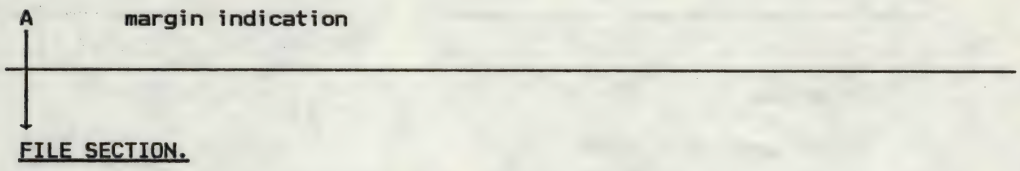
3.3 LANGUAGE ELEMENTS OF THE DATA DIVISION

3.3.1 FILE SECTION

Function

The FILE SECTION defines how the files are set up. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

Format



File description entries are discussed in the pages which follow. Record description entries are discussed in chapter 2 as well as in the pages which follow.

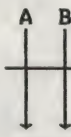
3.3.1.1 File Description (FD) Entry

Function

The file description (FD) entry specifies the physical setup, the identification, and the record names of a given file.

A file description entry must be written for each file to be processed by the program. The information contained in this entry generally pertains to the physical aspects of the file, that is, the description of the data as it appears on the input or output medium.

Format




```
A  B    margin indication
|  |
|  |
FD file-name

   [BLOCK CONTAINS clause]

   [RECORD clause]

   [RECORDING MODE clause]

   [LABEL RECORDS clause]

   [DATA RECORDS clause]

   [LINAGE clause]

   [CODE-SET clause].
```

Rules

1. The level indicator FD specifies the beginning of a file description entry and must precede the file-name.

2. The file-name must be identical to the file-name given in a SELECT clause.

3. The clauses which follow the file-name are optional except for the LABEL RECORDS clause. The order in which they appear is immaterial.

4. The file description entry is terminated by a period.

5. If the LABEL RECORDS clause is omitted, LABEL RECORD IS STANDARD is assumed.

6. Table 3-3 provides a summary of the functions of the clauses used in file description entries.

   The formats and functions of these clauses are described on the following pages.

Programming consideration

The file description entry must be followed by one or more record description entries.

For a discussion of the REPORT clause, see chapter 8, "Report Writer".

Clause	Function
BLOCK CONTAINS	Specifies block size (physical record size).
CODE-SET	Defines character code set for data output to external devices.
DATA RECORDS	Indicates the names of the data records in the file.
LABEL RECORDS	Gives the names and values of the label records contained in a file.
LINAGE	Specifies the size of a logical page. Also permits definition of a page heading and a page footing.
RECORD	Specifies logical record size.
RECORDING MODE clause	Specifies the format of the logical data records.

Table 3-2 Functions of file description clauses

BLOCK CONTAINS CLAUSE

Function

The BLOCK CONTAINS clause specifies the maximum size of a physical block.

Format

```
BLOCK CONTAINS [integer-1 TO] integer-2 {CHARACTERS}
                                        {RECORDS   }
```

Rules

1. integer-1 and integer-2 must both be at least 20 (for CHARACTERS, since 20 is the minimum block length required by the system) and must not contain more than 32763.

2. The CHARACTERS or RECORDS phrase indicates whether block length is to be specified as a multiple of characters or logical data records.

3. If neither the CHARACTERS or the RECORDS phrase is used, CHARACTERS will be assumed.

4. integer-1 TO integer-2 indicates the number of characters or data records in a given block, depending on the phrase used (either CHARACTERS or RECORDS).

5. When only integer-2 is specified, it refers to the maximum size of the block. When integer-1 and integer-2 are specified, they refer to the minimum and maximum size of the block, respectively. However, the phrase "integer-1", when specified, is used only for documentation purposes and is treated as comments by the compiler.

   The meaning of the **maximum block size** specified by this clause is as follows:

   The blocks of the file may not be longer but may be shorter than the specified length, as is frequently the case with unblocked or blocked data records whose length is variable.

6. When the CHARACTERS phrase is used, the block size is specified in terms of the number of characters contained within the block, regardless of the types of characters used to represent the data items within the block (see "USAGE Clause"). In this case, both integer-1 and integer-2 must include slack bytes and four characters for the record length field of each record of the block.

7. When the CHARACTERS phrase is used, and only integer-2 is specified, integer-2 indicates the exact length of the physical block. When both integer-1 and integer-2 are specified, they refer to the minimum or maximum physical block length, respectively.

8. When the RECORDS phrase is used, the block size is specified in terms of logical records. In this case, the compiler calculates the block size by multiplying the number of characters in the maximum size logical record by the value specified in integer-2, adding four characters for the block length field in the case of variable-length data records.

9. When the BLOCK CONTAINS clause is omitted, the compiler assumes that the records are not blocked, i.e. BLOCK CONTAINS 1 RECORDS is assumed. Consequently, the BLOCK CONTAINS clause may be omitted when **all** blocks of the file contain one, and only one, data record.

Programming considerations

1. Table 3-3 shows how the compiler calculates the block sizes in terms of characters, and the appropriate phrases contained in the BLOCK CONTAINS clause and RECORDING MODE clause.

   Legend for Table 3-4:

   F = fixed-length records
   V = variable-length records
   BL = block length
   SL = record length
   SL_{max} = maximum record length
   BLF = block length field (has the value 4)
   SLF = record length field (has the value 4)
   n = integer

Mode option of the RECORDING MODE clause	BLOCK CONTAINS [integer-1 TO] integer-2 CHARACTERS	RECORDS
F	BL = integer-2 (integer-2 = n*SL)	BL = integer-2*SL
V	BL = integer-2 + BLF	BL = integer-2* (SLmax + SLF) + BLF

Table 3-3 Calculation of the maximum block size

2. The CHARACTERS phrase should be used if the RECORDS phrase would imply too inaccurate a record size. For example, assume that the programmer describes a mode V Record of 100 characters, and that he writes a block consisting of a 50-character record followed by three 100-character records. If he specifies the clause BLOCK CONTAINS 4 RECORDS, the compiler will assume that the block length is 420 characters (i.e. 4 x (100 + 4) +4).

   However, since each block requires only 366 + 4 characters (i.e. 50+4) + 3 x (100+4) +4), the programmer should specify the following clause:

   BLOCK CONTAINS 366 CHARACTERS.

CODE-SET CLAUSE

Function

The CODE-SET clause specifies the character code set used to represent data on the external media.

Format

```
CODE-SET IS alphabet-name
```

Rules

1. When the CODE-SET clause is specified for a file,

   a) all data in that file must be described as USAGE IS DISPLAY and

   b) any signed numeric data must be described with the SIGN IS SEPARATE clause.

2. alphabet-name must not be a literal.

3. The CODE-SET clause may only be specified for non-disk-storage files.

Programming considerations

1. If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from/to the native character codes. This code conversion occurs during the execution of an input or output operation (see "SPECIAL-NAMES Paragraph").

2. If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media.

DATA RECORDS CLAUSE

Function

The DATA RECORDS clause is used only for documentation. It specifies the names of the data records in a file.

Format

```
DATA {RECORD  IS  } data-name-1 [data-name-2]...
     {RECORDS ARE}
```

Rules

1. data-name-1, data-name-2, etc. are the names of data records. Each of these records must be preceded by the level number 01 in the file description entry.

2. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, formats etc. The order in which they are listed is immaterial.

LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present, and identifies them if they are.

Format

```
LABEL { RECORD  IS } { OMITTED                          }
      {         }   { STANDARD                          }
      { RECORDS ARE } { data-name-1 [data-name-2]...    }
```

Rules

1. For data-name-1, data-name-2, etc., record description entries must be present for the file concerned. data-name-1, data-name-2, etc., must not appear as operands in the DATA RECORDS clause of this file. The data-name may optionally be used to specify, in the LINKAGE SECTION, a data record used for label handling.

2. The OMITTED phrase specifies that either there are no unique labels for this file or that the existing labels are non-standard and the user does not want handling by a USE procedure for label processing (e.g. he may want to process the labels as data records). OMITTED is prohibited in conjunction with UT TAPISO.

3. The STANDARD phrase specifies that labels are present for the file and that these labels are in accordance with system conventions (q.v. [2]).

4. In the following discussion, all references to data-name-1 likewise apply to data-name-2, etc.

   When data-name-1 is specified, this indicates either the presence of user labels in addition to standard labels, or the presence of non-standard labels. Data-name-1 defines the name of a user label data record.

   When processing user labels, data-name-1 may be defined for any files except unit-record (UR) files.

Programming considerations

1. The operand specified in the LABEL RECORDS clause depends on the class of equipment utilized, the organization of the data in the described file, and the type of device employed to hold the file. Table 3-5 summarizes these considerations and also indicates the user declaratives required when user labels (in conjunction with system labels) or nonstandard labels are to be processed (see also "USE Statement").

2. For the format of system labels, see under "System Standards" [1].

3. User labels are formatted as follows:

   a) Each user label is 80 characters long.

   b) Positions 1-3 of a user header label must contain the characters UHL.

   c) Positions 1-3 of a user trailer label must contain the characters UTL.

   d) Position 4 shows the relative location of the label in a sequence of header or trailer labels; that is, this position must contain a digit from 1 to 9. If only one label (UHL and/or UTL) label is present, position 4 must contain the character "1".

   e) Positions 5-80 are formatted according to the users specifications.

   For further details, see "System Standards" [1].

4. User header labels follow standard beginning file labels of the system, but precede the first data record.

5. User trailer labels follow standard closing file labels of the system.

6. Nonstandard labels can be from 1 to 4095 characters long. Their format and contents are defined by the user.

7. If data-name-1 is specified, then all Procedure Division references to the specified data-names, or to any items subordinate to these data-names, must appear within user declaratives (USE procedures).

Type of labels used		Labels omitted	Non-standard labels only	System labels only	System labels and user labels
Device classification	Clauses / Devices	LABEL RECORD IS OMITTED	LABEL RECORDS ARE file-name USE BEFORE	LABEL RECORDS IS STANDARD	LABEL RECORDS ARE file-name USE AFTER
UR	READER	X			
	PNRINTER	X			
	PRINTERANS	X			
	PRINTERDOD	X			
	SYSLST			X	
UT	TAPE	X	X	X	X
	TAPISO		X	X	X
	TAPDISC	X	X	X	X
	DISC			X	X
DA	DISC			X	X

Table 3-4 LABEL RECORDS clause considerations

LINAGE CLAUSE

Function

The LINAGE clause provides a means of specifying, for an output file, the size of a logical page in terms of the number of lines. It can also be used to specify the size of the top and bottom margins on the logical page and the line within the page body at which the footing area is to begin. The LINAGE clause is only allowed in connection with PRINTERANS.

Format

```
LINAGE IS {data-name-1 } LINES
          {integer-1   }

[WITH FOOTING AT {data-name-2}]
                 {integer-2  }

[LINES AT TOP {data-name-3}]
              {integer-3  }

[LINES AT BOTTOM {data-name-4}]
                 {integer-4  }
```

Rules

1. data-name-1, data-name-2, data-name-3 and data-name-4 must be unsigned numeric and integer data items.

2. The value of integer-1 or of the data item referenced by data-name-1 must be greater than zero.

3. The value of integer-2 or of the data item entered under data-name-2 must be greater than zero, but must not be greater than integer-1 on the data item referenced by data-name-1.

4. The value of integer-3 and integer-4, or the data items referenced by data-name-3 and data-name-4, may be 0.

5. The LINAGE clause is only permissible for PRINTERANS. It provides a means of specifying the size of a logical page in terms of the number of lines. The logical page size is the sum of the values of all entries except the FOOTING entry. If the LINES AT TOP or LINES AT BOTTOM is not specified, the value for this function is zero. If the FOOTING phrase is not specified, the assumed value is equal to integer-1, or the contents of the data item data-name-1, whichever is specified. (See Table 3-6).

top margin LINES AT TOP	data-name-3/ integer-3	≥ 0 (default = 0)
page body LINAGE IS	data-name-1/ integer-1	1 2 3 . . . n — Number of print lines
footing area WITH FOOTING AT		data-name-2/ (default = 1 footing line) integer-2 . . . data-name-1/ integer-1
bottom margin LINES AT BOTTOM	data-name-4/ integer-4	≥ 0 (default = 0)

Table 3-5 Setup of a logical page

6. The size of a logical page must not necessarily correspond to the size of a physical page.

7. The value of integer-1 or the data item referenced by data-name-1 specifies the number of lines that can be written and/or spaced on the logical page. This part of the logical page, in which these lines can be written and/or spaced, is called the page body.

8. The value of integer-3 or data item data-name-3 specifies the number of lines forming the top margin of the logical page. This area is left blank.

9. The value of integer-4 or data item data-name-4 specifies the number of lines forming the bottom margin of the logical page. This area is left blank.

10. The value of integer-2 or data item data-name-2 specifies the line number within the page body at which the footing area begins.

11. The footing area comprises the area of the logical page between the line specified by the value of integer-2 or data-name-2 and the line specified by the value of integer-1 or data item data-name-1 (integer-1, integer-2).

12. The values of integer-1, integer-3 and integer-4 (or the values of data items data-name-1, data-name-3 and data-name-4, if specified) will be used at object time by an OPEN statement with OUTPUT phrase, to specify the number of lines in each of the indicated parts of the first logical page. The value of integer-2 or of data item data-name-2 (if specified) will be used at that time to define the footing area.

If page overflow occurs when a WRITE statement with ADVANCING phrase is being executed, the values of integer-1, integer-3 and integer-4, or of the data items data-name-1, data-name-3 and data-name-4, will be used to specify the number of lines that are to comprise the indicated parts of the next logical page.

The value of integer-2 or of data item data-name-2 (if specified) will be used to define the footing area of the next logical page.

Programming considerations

1. The COBOL register LINAGE-COUNTER is generated whenever a LINAGE clause occurs. The LINAGE-COUNTER value at any given time represents the line number at which the printer is positioned within the current page body. The first printable line on each logical page has the number 1.

   A separate LINAGE-COUNTER is supplied for each file described in the FILE SECTION whose file description entry contains a LINAGE clause.

2. The LINAGE-COUNTER may be accessed, but may not be modified, by Procedure Division statements. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.

3. The LINAGE-COUNTER is automatically updated during the execution of a WRITE statement for the file:

   a) If the ADVANCING PAGE phrase is specified in the WRITE statement, the LINAGE-COUNTER is automatically reset to the value 1.

   b) If ADVANCING integer or ADVANCING identifier-2 is specified in the WRITE statement, the LINAGE-COUNTER is incremented by integer or by the value of data item identifier-2.

   c) If the ADVANCING phrase is omitted in the WRITE statement, the LINAGE-COUNTER is incremented by the value 1.

   d) The LINAGE-COUNTER is automatically reset to the value 1 when the printer is positioned to the first line to be written on the next logical page. (See WRITE Statement.)

   e) The LINAGE-COUNTER is automatically set to the value 1 at the time an OPEN statement is executed for the file.

RECORD CLAUSE

Function

The RECORD clause defines the length of the records in a file and controls the external data record format (see RECORDING MODE clause).

Format 1 Indicates fixed-length records by specifying the number of character positions in a record.

Format 2 Indicates variable-length records, whereby the size of the record must lie within a certain range.

Format 3 Indicates variable-length records, whereby the minimum and maximum number of character positions are specified.

Format 1

```
RECORD CONTAINS integer-1 CHARACTERS
```

Rules

1. The length of each data record is precisely defined by its data description entry. The length specification in the RECORD CONTAINS clause is ignored.

2. The number of character positions in each record description entry must be equal to integer-1.

3. integer-1 must be at least 1 and can be as large as 32767.

Format 2

```
RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]
    [DEPENDING ON data-name-1]
```

Rules

1. In any record description entry, it is impermissible for the length specification to be less than integer-2 or more than integer-3.

2. integer-3 must be larger than integer-2.

3. integer-2 must be at least 1, while integer-3 can be up to 32763.

4. data-name-1 must be described as an unsigned integer data item in the WORKING-STORAGE or LINKAGE section.

5. The length of a record is determined by the sum of the character positions of all its constituent data items and the slack bytes created by the compiler. Data description entries described with the REDEFINES or RENAMES clause are not counted when calculating the length. If the record contains a table, the minimum or maximum number of table items is taken into account when calculating the length (for further details see SYNCHRONIZED and USAGE clauses as well as "Data Alignment").

6. If integer-2 is omitted, it is assumed that the length of the shortest record is 1.

7. If integer-3 is omitted, the length of the longest record in the record description entry for this file is assumed. This number also provides the block size.

8. If data-name-1 is specified, the number of character positions in the record must be moved to data-name-1 before a RELEASE, REWRITE or WRITE statement is executed for this file.

9. If data-name-1 is specified, its contents remain unchanged when a RELEASE, REWRITE or WRITE statement is executed or when a READ or RETURN statement terminates abnormally.

10. During execution of a RELEASE, REWRITE or WRITE statement, the record length is determined as follows:

    a) By the contents of data-name-1, if specified.

    b) By the name of character positions in the record if data-name-1 is omitted and the record description entry has no OCCURS clause with DEPENDING ON phrase.

    c) By the fixed portion of the record description entry (i.e. all data description entries without OCCURS DEPENDING ON) and by the number of recurrences of the table items during execution when data-name-1 is omitted and the record description entry has no OCCURS clause with DEPENDING ON phrase.

11. If data-name-1 is specified and a READ statement has been successfully executed, data-name-1 contains the number fo character positions of the record just read. This number is, however, not greater than integer-3, but it may be larger than the largest record description entry.

12. If INTO is specified in a READ statement, the number of character positions in the current record, which is the sending field in the implicit MOVE, is determined as follows:

    a) By the contents of data-name-1, if specified.

    b) By the value that would be transferred if data-name-1 had been specified.

Format 3

RECORD CONTAINS integer-4 TO integer-5 CHARACTERS

Rules

1. integer-4 describes the number of character positions in the shortest record, integer-5 the number in the longest.

2. The length of a record is determined by the sum of the character positions in all elementary items and by the slack bytes created by the compiler. Data description entries described with the REDEFINES or RENAMES clause are not counted in the length calculation. If the record contains a table, the minimum or maximum number of table items is taken into account in the length calculation. (For further information see the SYNCHRONIZED and USAGE clause as well as "Data Alignment").

3. integer-5 cannot exceed 32763.

RECORDING MODE CLAUSE

Function

The RECORDING MODE clause specifies the format of logical records in a file.

Format

RECORDING MODE IS mode

Rules

1. mode designates the record format, and can assume the values F, V or U.

2. F

   Indicates fixed-length records. It can only be specified when all logical records in the file have the same length. When F is used as the mode entry, the following rules apply:

   a) No record description entry in the file is permitted to contain an OCCURS clause with DEPENDING ON phrase.

   b) Blocks may contain more than one record, in which case a fixed number of records per block is implied.

   c) If the file description (FD) entry is followed by more than one record description entry, then all records in this file must have the same length.

   d) No block length fields (BLF) or record length fields (SLF) are used in F-mode blocks and records.

3. V

   Refers to variable-length records. It may be used to describe variable- or fixed-length records. When V is used as the mode entry, the following rules apply:

   a) Blocks may contain more than one record; however, all records specified for a block must be wholly contained within the block.

   b) Each V-mode record is preceded by a control field containing the length of the logical record (record length field, SLF). A block of variable-length records also contains a block descriptor control field (block length field, BLF). These control fields must not be described in Data Division record descriptions, since the compiler automatically provides for them. These fields are not available to the user.

4. U

Refers to records in undefined format. It may be used for any combination of record descriptions (either fixed or variable). Mode U is comparable to mode V, except that U-mode records may not be blocked and have no preceding control field (record length field). When mode U is specified, the BLOCK CONTAINS clause is not required.

Programming considerations

If the RECORDING MODE clause is not explicitly written for an FD entry, the recording mode of the file is determined by the RECORD clause as shown in Table 3-6.

a) If the RECORD clause is not specified , the recording mode assumed for the file is V (variable).

b) If the RECORD CONTAINS clause specifies only one number (i.e. RECORD CONTAINS integer-1 CHARACTERS), the recording mode assumed for the file is F (fixed).

c) If the RECORD CONTAINS clause specifies two numbers (i.e. RECORD CONTAINS integer-4 TO integer-5 CHARACTERS), the recording mode assumed for the file is V (variable).

d) If the RECORD IS VARYING IN SIZE clause is specified, variable-length records are assumed.

e) If the ASSIGN clause was used to assign the symbolic device name SYSLST and RECORDING MODE F or V was specified, then the RECORDING MODE clause is regarded as comments and RECORDING MODE IS V is assumed.

Mode option in the RECORDING MODE clause	Phrase in the RECORD clause: RECORD CONTAINS Clause not written	integer-1 CHARACTERS	integer-4 TO integer-5 CHARACTERS	RECORD IS VARYING IN SIZE
Clause not specified	V	F	V	V
F	F	F	V1)	V1)
V	V	V 1)	V	V
U	U	U	U	U

Table 3-6 Relationship between the RECORD and RECORDING MODE clauses

1) The entries in the two clauses are contradictory; V is assumed.

3.4 LANGUAGE ELEMENTS OF THE PROCEDURE DIVISION

3.4.1 Input/Output Statements

In COBOL, input and output is record-oriented. Thus the READ, WRITE and REWRITE statements process records. The COBOL user is therefore only concerned with the processing of individual records. The following operations are performed automatically: moving data into input/output areas (buffers) and/or internal storage, validity checking, correcting errors (where feasible), blocking/unblocking, and switching volumes.

Note

The description of input/output statements uses the expressions "volume" and "reel". Volume may be used for all input/output devices; reel is applicable to magnetic-type devices only. The handling of disk-storage files with sequential access is logically equivalent to the handling of magnetic-tape files.

Summary

Statement	Function
CLOSE	Terminates processing of a reel, unit or file.
OPEN	Opens a file for processing.
READ	Makes a record available from an input or input/output file.
REWRITE	Replaces a data record on a disk-storage file by a specified data record.
USE	In addition to input/output statements, the USE statement may be supplied to indicate label and error handling procedures (see "Declaratives Subdivision of the Procedure Division").
WRITE	Releases a record to an input/output or output file.

CLOSE STATEMENT

Function

The CLOSE statement terminates the processing of input/output reels, units, and files, with optional rewind and/or lock where applicable.

Format

```
CLOSE file-name-1 [[{REEL}] [[WITH {NO REWIND}]]]
                  [{UNIT}]  [{     {LOCK     }]]
                            [{FOR   REMOVAL  }]]

    [file-name-2 [[[{REEL}] [[WITH {NO REWIND}]]]]]
                   [{UNIT}]  [{     {LOCK     }]]
                             [{FOR   REMOVAL  }]]]]
```

Rules

1. file-name-1, data-name-2,... specify files on which the CLOSE statement is to operate.

2. data-name-1, data-name-2,... must be described in a file description entry in the Data Division of the COBOL program.

3. The files referenced in the CLOSE statement need not all have the same organization or access.

4. The following descriptions are only definitions of the terms used in the statement format. Complete details on the use of each phrase are given further below.

REEL	indicates that the current tape reel should be closed.
UNIT	indicates that the current mass-storage unit should be closed.
NO REWIND	indicates that a tape file should not be repositioned to the beginning of the reel after it has been closed.
LOCK	indicates that the file cannot be reopened in the same program run.
REMOVAL	indicates that the current reel of a magnetic tape file is to be unloaded.

5. If not specified otherwise, the terms REEL and UNIT are equivalent and are fully interchangeable within a CLOSE statement. The handling of sequential disk-storage files is logically equivalent to the handling of files on magnetic tape or similar sequential media.

6. A CLOSE statement may be executed only on an open file.

7. In order to show the effect of various CLOSE phrases as applied to various storage media, all input/output files are divided into the following categories:

   a) **UNIT RECORD volume file:**
   a file assigned to an input or output medium for which the concepts rewinding, units, and reels have no meaning.

   b) **Sequential single volume file:**
   a sequential file that is entirely contained on one volume. There may be more than one file on this volume.

   c) **Sequential multivolume file:**
   a sequential file contained on more than one volume.

8. The following paragraphs describe the meanings of the various CLOSE statements in sequential file processing.

   The results of executing each CLOSE phrase for each type of file are summarized in Table 3-9. The definitions of the symbols used in it are given below. When the definition of the symbol depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to files opened as INPUT, OUTPUT and I-O.

   A - **Previous volumes unaffected**

   All volumes in the file prior to the current volume are processed according to standard volume switching procedures, except those volumes controlled by a prior CLOSE REEL/UNIT statement.

   B - **No rewind of current reel**

   The current volume is positioned at the logical end of the file on the volume.

   C - **Standard close file**

   For files opened as INPUT or I-O: if the file is positioned at its end and a LABEL RECORDS clause was supplied, then (if a USE procedure is present) the standard trailer label procedure and user trailer label procedure are executed. The order in which these two routines are executed is specified by the USE procedure. The standard system closing procedures are then performed.

   If the file is positioned at its end, but a LABEL RECORDS clause was not supplied, then only the standard system closing procedures are performed.

   If the file is positioned other than at its end, only the standard system closing procedures are performed. Even if USE procedures are supplied, no label processing will take place in this case. (An INPUT or I-O file is considered to be at its end if the imperative statement in the AT END phrase of the READ statement, if entered, has been executed, and no CLOSE statement has been executed.)

   For files opened as OUTPUT:
   if a LABEL RECORDS clause was specified for this file, the standard ending label procedures and the user ending label procedures (provided such were specified by a USE procedure are performed).

The order in which these two procedures are executed is defined by the USE statement. The standard system closing procedures are then performed.

If label records are not specified for the file, only the standard system closing procedures are performed.

D - **Unload current reel**

The REMOVAL option, when specified, causes the current reel of a magnetic tape file to be unloaded. Further processing of the file requires an appropriate continuation reel. After executing a CLOSE statement without the REEL/UNIT phrase specified, and an OPEN statement for this file, this reel may be processed again.

E - **Standard file lock**

If the LOCK phrase is used, the file cannot be reopened in the same program run.

F - **Standard close volume**

For files opened as INPUT or I-O, the following operations are carried out:

a) Volume swapping.

b) USE procedures for processing standard and user header labels (if specified by a USE statement). The order in which these procedures are executed is defined by the USE statement.

c) The next data record on the new volume is made available for the subsequent read operation.

For files opened as OUTPUT, the following operations are carried out:

a) Standard and user trailer label procedures (if specified by a USE statement). The order in which these procedures are executed is defined by the USE statement.

b) Volume swapping.

c) Standard and user header label procedures (if specified by a USE statement). The order in which these procedures are executed is defined by the USE statement.

G - **Rewind**

The current volume is positioned at its beginning (for magnetic tape file, this is the beginning of the reel; for disk-storage files, this is the beginning of the file concerned).

X - **Illegal**

This is an illegal combination of a CLOSE phrase and type of file.

CLOSE phrase	File type		
	Unit record	Sequential single-volume	Sequential multivolume
CLOSE	C	C, G	C, G, A
CLOSE WITH LOCK	C, E	C, G, E	C, G, E, A
CLOSE WITH NO REWIND	X	C, B	C, B, A
CLOSE FOR REMOVAL	X	G, D	F, G, D
CLOSE REEL/UNIT	X	X	F, G
CLOSE REEL/UNIT FOR REMOVAL	X	X	F, G, D
CLOSE REEL/UNIT WITH NO REWIND	X	X	F, B

Table 3-7 Relationship of types of sequential files and the options of the CLOSE statement

General note:
If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, then the next input/output statement to be executed for that file (if any) must be an OPEN statement.

9. The Execution of a CLOSE statement causes the contents of the data item specified in the FILE STATUS clause to be updated (see also "FILE STATUS clause").

OPEN STATEMENT

Function

The OPEN statement open files for processing and performs label checking and output.

Format

```
                ┌INPUT  {file-name  [REVERSED      ]}...┐
OPEN            │                   [WITH NO REWIND]  │
                │OUTPUT {file-name [WITH NO REWIND]}...  │...
                │I-O    {file-name}...                   │
                └EXTEND {file-name}...                   ┘
```

Rules

1. The file-name must be defined by a file description (FD) entry in the Data Division.

2. At least one of the phrases INPUT, OUTPUT, I-O, or EXTEND must be used. If two or more such phrases are used, they may appear in any order. Two or more file-names may be supplied for one option. However, the same file-name must not appear more than once in a given OPEN statement.

3. The I-O phrase is permitted for disk-storage files only.

4. The files specified in a given OPEN statement may have different organizations or access modes.

5. The EXTEND phrase is prohibited for device class UR.

6. The EXTEND phrase must not be used for multi-file volumes (see also "I-O-CONTROL Paragraph").

7. After successful execution of an OPEN statement, the availability of the file specified by file-name is established, and the file is in the open mode.

8. After successful execution of an OPEN statement, the associated record area is available to the program.

9. Before the first successful execution of an OPEN statement for a file, no statement (other than a SORT/MERGE statement with the USING or GIVING phrase) must be executed that would either explicitly or implicitly make reference to that file.

10. INPUT means that the file is to be processed as an input file (input mode).

11. OUTPUT indicates that the file is to be processed as an output file (output mode).

12. I-O indicates that input and output operations (that is, reading, writing, and updating operations) are to be performed on the file.

    Since this phrase implies the existence of the file, it cannot be used if the disk-storage file is being initially created (update mode).

13. The EXTEND phrase, when specified, causes the file to be positioned immediately behind the last logical data record of the file. By subsequent WRITE statements for that file, data records will be added to the file in the same manner as if it had been opened with the OUTPUT phrase specified (extend mode).

14. REVERSED indicates that records in the file are to be read in reversed order, i.e. starting with the last record of the file.

    The REVERSED phrase can be used only with a sequential single-reel file.

    The REVERSED phrase should not be specified for a file with nonstandard labels. In all other cases, the system will read label records as if they were data records.

    The REVERSED phrase may be specified for a file written on a 7-level tape utilizing the PACK/UNPACK feature, only if the length of each block in that file is a multiple of three.

15. NO REWIND may be specified for a tape file or for a disk-storage file which was opened OUTPUT. The actions taken for tape and disk-storage files are listed below:

    Tape files:

    NO REWIND indicates that the tape reel is not to be rewound when the file is opened (see Rule 14).

    Disk-storage files:

    NO REWIND indicates that no records existing on the file are to be overwritten. NO REWIND exists only for reasons of compatibility. OPEN OUTPUT WITH NO REWIND has the same function as OPEN EXTEND.

16. If the NO REWIND phrase is omitted, the file is assumed to be a work file; the records on the file are considered transient and may be overwritten. In this case, all tracks are released and are written on by any subsequent WRITE statements. This capability permits the use of disk-storage files for temporary storage and obviates the need for releasing and purging a file before it may be assigned as an output file.

17. If the storage medium containing the file permits the rewind function, the following rules apply:

    a) If neither the REVERSED, nor the EXTEND, nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

    b) If the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; instead, the file is expected to be at its beginning already.

    c) If the REVERSED phrase is specified, the data records of the file are made available in reversed order; that is, starting with the last data record of the file.

18. Label handling procedures are performed by the OPEN statement for all files, except those for which LABEL RECORDS ARE OMITTED is specified in the file description entry.

    If the INPUT phrase of the OPEN statement is specified, the following steps are taken:

    a) If system labels are present, they are checked.

    b) If user labels or nonstandard labels are present and an applicable USE procedure is declared, that USE procedure is executed.

    c) The file is positioned so that the first record can be read.

    If the OUTPUT phrase of the OPEN statement is supplied, execution takes place as follows:

    a) If standard labels are specified, they are created and written.

    b) If an applicable USE procedure is declared, it is executed; and user or nonstandard labels are written.

    c) The record area is made available to the program, to receive data.

    If the I-O phrase of the OPEN statement is specified, the following steps are taken:

    a) If system labels are present, they are checked.

    b) If user labels are present and an applicable USE procedure is declared, that USE procedure is executed.

    c) The file is positioned for data read or write operations.

    If the EXTEND phrase as well as the LABEL RECORDS STANDARD/data-name clause are specified for an OPEN statement, then execution of the OPEN statement involves the following steps:

    a) The file header labels are processed only if it is a single-reel or single-volume file.

    b) The reel/volume header labels on the last existing reel/volume are processed as in the case of the OPEN statement with INPUT phrase.

    c) The existing file trailer labels are processed as in the case of the OPEN statement with INPUT phrase. These label records are then deleted.

    d) Further processing is then the same as with OPEN OUTPUT.

19. After execution of an OPEN statement, the contents of the data-name in the FILE STATUS clause (if specified) will be updated (see also "FILE STATUS Clause").

Programming considerations

1. A unit-record file must only be opened either as an input or an output file, depending on the device the file was assigned to; thus, for example, a file assigned to a card reader must be opened only as an input file (INPUT phrase specified in the OPEN statement). Utility files may be opened INPUT as well as OUTPUT.

2. All of the phrases INPUT, OUTPUT, I-O or EXTEND may be used, within the same program, in different OPEN statements for a given file. Before executing another OPEN statement after the logically first one for the same file in a program, a CLOSE statement must have been performed; at this time, REEL/UNIT or LOCK must not be specified in the CLOSE statement.

3. Table 3-8 lists the statements permissible in OPEN mode (X indicates "permitted"):

Statement	OPEN mode			
	Input	Output	Input-Output	Extend
READ	X		X	
REWRITE			X	
WRITE		X	X	X

Table 3-8 Valid input-output statements in each OPEN mode

4. Disk-storage files may be opened INPUT, OUTPUT, I-O or EXTEND.

5. If OPTIONAL was specified in the SELECT clause for an input file, a check will be made at object time for the existence of that file. If the file does not exist, execution of the first READ statement will cause the AT END phrase to occur.

READ STATEMENT

Function

When the READ statement is used in sequential file processing, the next data record from a file is made available to the program.

```
READ file-name [NEXT] RECORD [INTO identifier]
               [AT END imperative-statement.]
```

Format 2 see chapters 4 and 5.

Rules

1. file-name must be defined by a file description entry in the Data Division of the program.

2. The NEXT entry is optional and has no effect on the execution of a READ statement.

3. If no USE procedure is declared for the file, AT END must be supplied in the READ statement.

4. The READ statement makes available to the program a record from the file specified by file-name, unless an exceptional condition exists. Here, the data record is read into the input area which was specified by the record description entry in the file description. It remains available there until the next input/output statement is executed for that file.

5. A READ statement with INTO phrase specified is equivalent to the following rules, which are executed in the order given below:

   a) The same READ statement is executed without INTO phrase.

   b) The current data record is moved from the input area to the area specified by identifier (implicit MOVE). The move operation takes place according to the rules for MOVE statements without CORRESPONDING phrase. After execution of the READ statement with INTO phrase, the record is available both in the input area and the are specified by identifier. The length of the sending field is determined by the length of the record entered (see RECORD clause).

      If the file description entry contains a RECORD clause with VARYING phrase, the implicit move applies to a group item.

      If the READ statement described in a) was successful, no move operation takes place.

      If the input area is to be referenced following a READ statement without INTO phrase, the user is responsible for using the correct record description (i.e. corresponding to the length of the input record).

6. If a file for which OPTIONAL was supplied in the SELECT clause does not exist at object time, the first READ statement for that file will transfer control to the statements specified in the AT END phrase.

7. If the physical end of a reel or disk-storage volume is encountered during execution of a READ statement for a multi-volume file, then the following operations take place:

   a) The standard ending volume label procedure and, if specified by an appropriate USE procedure, the user's ending volume label procedure are executed. The order in which the two procedures are performed is specified by the USE procedure.

   b) A volume switch is executed.

   c) The standard beginning volume label procedure and, if specified, the user's beginning volume label procedure are executed. Again, the order in which these procedures are executed is specified by the USE procedure.

8. Execution of a READ statement causes the contents of the data item specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS Clause").

9. Following an unsuccessful attempt to perform a READ statement, the contents of the input area associated with the file as well as the position within the file are unpredictable.

10. If the number of character positions in a read data record is greater than the largest record description, the record will be truncated accordingly; if the number is less than the smallest record description, the contents to the right of the last valid character will be unpredictable (for the length of a record description see "RECORD Clause"). In either case the read operation is successful, but a FILE STATUS is set to indicate the occurrence of a record length conflict.

Programming considerations

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.

2. When a file contains more than one logical record type, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information in the current record is accessible. Thus, no statement in the Procedure Division can refer to a different logical record. Similarly, no reference is permitted to the nth occurrence of data in a data record that occurs fewer than n times (see the description of the DEPENDING ON phrase, under the OCCURS clause). If either type of reference is attempted, the results will be unpredictable.

3. If a READ statement is executed, and there is no further data record in the file, then an end-of-file condition occurs, and the execution of that READ statement is considered unsuccessful (see "FILE STATUS Clause").

4. If the end-of-file condition is encountered, the following steps are performed, in the order given below:

   a) The contents of the data item from the FILE STATUS clause, if specified for the file, is supplied with a value indicating the AT END condition (see "FILE STATUS Clause").

b) If AT END is specified in the READ statement, control is transferred to the imperative statement following the AT END phrase. Any USE statement declared for that file will **not** be executed.

c) If the AT END phrase is omitted from the READ statement, either an explicit or implicit USE procedure must be declared for the file. When the AT END condition occurs, that USE procedure will then be executed.

When the AT END condition occurs, execution of the input/output statement must be considered to be unsuccessful.

5. After the AT END condition has occurred, no format-1 READ statement may be issued for the file until a successful close statement has been carried out, followed by a successful OPEN statement for that file.

REWRITE STATEMENT

Function

The REWRITE statement replaces a logical record on a disk-storage file.

Format

```
REWRITE record-name [FROM identifier]
```

Rules

1. record-name identifies the record to be replaced.

2. record-name and identifier must not refer to the same storage area.

3. record-name must be defined in a file description entry in the Data Division of the program.

4. Execution of a REWRITE statement with the FROM phrase specified corresponds to the following statements:

   ```
   MOVE identifier TO record-name
   REWRITE record-name.
   ```

   The contents of the storage area described by record-name will not effect the execution of the REWRITE statement until the implicit MOVE statement has been carried out.

   Therefore, when using the FROM phrase, data is transferred from identifier to record-name and is then released to the appropriate file. Identifier may be used for referencing any data outside the current file description entry.

   Transfer of data through the implicit MOVE statement takes place in accordance with the rules for a MOVE statement without the CORRESPONDING phrase. After the REWRITE statement is executed, the contents of the record is still available in the area specified by the "identifier" although it is no longer available in the area specified by record-name.

5. A REWRITE statement must be preceded by a successful READ statement as the last input/output statement for the associated file.

6. Execution of the REWRITE statement causes the record accessed by the preceding READ statement to be replaced on that file.

7. The data record rewritten by a successful REWRITE statement is no longer accessible in the record area; an exception to this rule is the use of the SAME RECORD AREA clause. In this case the data record is available to all other files specified in the SAME RECORD AREA clause as well as to the current file.

8. Execution of a REWRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS clause").

Programming considerations

1. The file associated with record-name must be a disk-storage file and must have been opened I-O at the time the REWRITE statement is executed.

2. The data record addressed by the data record name must be of the same length as the data record to be replaced on the file.

3. Rewriting a data record does not cause the record contents on the associated disk-storage file to be changed until the next block of the file is read or that file is closed.

USE STATEMENT

Function

The USE statement introduces declarative procedures and defines the conditions for their execution. The USE statement itself, however, is not executed.

Format 1 declares label handling routines.

Format 2 declares procedures to be run if an input/output error occurs in a file.

Format 1

```
USE [BEFORE]          [ENDING   ]  [REEL]
    {      } STANDARD [         ]  [UNIT]
    [AFTER ]          [BEGINNING]  [FILE]

                        [file-name-1 [file-name-2]...]
                        [OUTPUT                      ]
    LABEL PROCEDURE ON  {INPUT                       }
                        [I-O                         ]
                        [EXTEND                      ]
```

Rules for format 1

1. file-name must be defined in a file description (FD) entry of the program's Data Division.
2. file-name must not be the name of a sort-file.
3. file-name refers to the file description entry for which the specified label handling procedures are to be performed.
4. The procedures specified are executed in conjunction with OPEN and CLOSE statements for the file.
5. Once a USE procedure has been executed, program execution resumes with the calling routine.
6. INPUT indicates that the specified procedures are to be performed only on files which are in the input mode (OPEN statement with INPUT phrase).
7. OUTPUT indicates that the specified procedures are to be performed only on files which are in the output mode (OPEN statement with OUTPUT phrase).
8. I-O indicates that the specified procedures are to be performed only on files which are in the update mode (OPEN statement with I-O phrase).
9. EXTEND indicates that the specified procedures are to be performed only on files which are in the extend mode (OPEN statement with EXTEND phrase).

10. BEFORE or AFTER indicates the types of labels to be processed.

    BEFORE indicates that nonstandard labels are to be processed (such labels may be specified only for tape files).

    AFTER indicates that user labels are to following standard file labels, and that the user labels are to be processed.

11. BEGINNING or ENDING indicates that header or trailer labels, respectively, are to be processed.

    If the BEGINNING and ENDING phrases are omitted, the declared procedures will be run for both header and trailer labels.

12. The phrases REEL, UNIT or FILE indicate that the declared procedures are to be run when volume, reel or file labels are encountered.

    The REEL phrase is not applicable to disk-storage files. The UNIT phrase is not applicable to files in the random-access mode, since only file labels are processed in that mode. The compiler treats the REEL and UNIT phrases as interchangeable.

13. If the above phrases are omitted, the declarative procedures are executed depending on the type of volume involved, either for reel labels and file labels, or for volume labels and file labels.

Programming considerations for format 1

1. The labels to be processed for a file must be specified within the file description entry of that file, as data-names in the LABEL RECORDS clause, and must be defined as level-01 data items within the file description entry or the LINKAGE SECTION.

2. The same file-name may appear in more than one variant of the format-1 USE statement. However, this must not cause two or more declarative procedures to be initiated at the same time.

3. If the file-name phrase is used, the file description for the file-name must not specify a LABEL RECORDS clause with OMITTED phrase.

4. If the INPUT, OUTPUT or I-O phrases are specified, the declarative procedures do not apply to any input, output or update files, respectively, that are described with the LABEL RECORDS clause with OMITTED phrase.

5. The standard system procedures are performed on all standard label records consisting of system and user labels.

    a) Labels on input or input/output files are checked in the following order:

        1. The I-O system checks standard header labels.

        2. The USE procedures (if any) check user header labels.

        3. The I-O system checks standard trailer labels.

        4. The USE procedures (if any) check user trailer labels.

b) Labels on output files are created in the following order:

1. The I-O system created standard header labels.
2. The USE procedures (if any) create user header labels.
3. Before the user header labels are written, they are checked to see whether they begin with the string UHL.
4. I-O system creates standard trailer labels.
5. The USE procedures (if any) create user trailer labels.
6. Before the user trailer labels are written, they are checked to see whether they begin with the string UHL.

6. If the INPUT, OUTPUT or I-O phrase is specified in a USE procedure, the associated declarative procedures may contain unqualified references to what are called "common label items". A common label item is an elementary item that appears in each label record of the program, and has a common position and description within each label record. The common label item must not appear in any data record of the program (see also example 3-4).

7. Within a USE procedure, there must be no reference to non-declarative procedures except for the PERFORM * statement.

   Reference to procedure names which are subordinate to a USE statement may be made from another procedure, or non-declarative procedure, only by using a PERFORM statement.

8. The exit from a format-1 declarative procedure is generated by the compiler following the last statement in a given section. All logical paths within that section must lead to this exit point.

9. There is one exception to Programming consideration 8:
   The statement GO TO with the MORE-LABELS phrase may be used as a special exit point. After this statement is executed, the runtime system will take one of the following actions:

   a) If labels are currently being created, the system will write the current header or trailer labels; the program will then continue at the beginning of the declarative section in order to create more labels. The user must make sure that the last statement in the section is executed after all labels are created. At this point, it should be noted that, after execution of the GO TO statemOnt with the MORE-LABELS phrase, a label will be written in any case; if the user did not supply a new label, the runtime system will generate a dummy label record applying Programming consideration 8 accordingly.

   b) If labels are currently being checked, the system will read an additional header or trailer label; the program will then continue at the beginning of the declarative section in order to check more labels. However, when processing user labels, the system will reenter the section only when there is another user label to check. Thus, in this case, the programmer does not have to provide a program path that flows through the last statement in the section. On the other hand, when processing nonstandard labels, the system does not know how many labels exist. Therefore, the last statement in the section must be executed, in this case, to terminate nonstandard label processing.

Example 3-3 (format 1)

In this example, one declarative section (ALPHA) handles header labels and another (BETA) handles trailer labels. Note that the declarative procedures are executed both in input and output modes.

Data Division entries:

```
     FD  SAMPLE-FILE,
         RECORDING MODE IS F,
         BLOCK CONTAINS 5 RECORDS,
         RECORD CONTAINS 100 CHARACTERS,
         LABEL RECORD IS SAMPLE-LABEL,
         DATA RECORD IS SAMPLE-RECORD.
     01  SAMPLE-LABEL.
         02  LABEL-ID                PICTURE X(4).
         02  LABEL-INFO              PICTURE X(76).
     01  SAMPLE-RECORD               PICTURE X(100).
     WORKING-STORAGE SECTION.
     77  I-O-INDICATOR               PICTURE X.
         88  INPUT-MODE              VALUE "I".
         88  OUTPUT-MODE             VALUE "O".
     77  LABEL-COUNTER
         .
         .
         .
```

Procedure Division statements:

```
     DECLARATIVES.
     ALPHA-SECTION. USE AFTER STANDARD BEGINNING FILE LABEL PROCEDURE ON
        SAMPLE-FILE.
     ALPHA-1.
         IF INPUT-MODE THEN GO TO ALPHA-3.
         IF LABEL-COUNT = 0 THEN NEXT SENTENCE ELSE GO TO ALPHA-2.
         MOVE "UHL1" TO LABEL-ID.
         MOVE "THIS WAS PRODUCED BY ALPHA SECTION." TO LABEL-INFO.
         DISPLAY "511030 THIS LABEL CREATED:-" SAMPLE-LABEL.
         MOVE 1 TO LABEL-COUNT.
         GO TO MORE-LABELS.
     ALPHA-2.
         MOVE "UHL2" TO LABEL-ID.
         MOVE "THIS WAS THE SECOND LABEL PRODUCED BY ALPHA SECTION."
            TO LABEL-INFO.
         DISPLAY "511530 AND THIS LABEL TOO:-" SAMPLE-LABEL.
         MOVE 2 TO LABEL-COUNT.
         GO TO ALPHA-END.
     ALPHA-3.
         DISPLAY "512010 THIS LABEL READ:-" SAMPLE-LABEL.
         ADD 1 TO LABEL-COUNT.
         GO TO MORE-LABELS.
     ALPHA-END.EXIT.
     BETA SECTION. USE AFTER STANDARD ENDING FILE LABEL PROCEDURE ON
        SAMPLE-FILE.
     BETA-1.
         IF INPUT-MODE THEN GO TO BETA-3.
         IF LABEL-COUNT = 0 THEN NEXT SENTENCE ELSE GO TO BETA-2.
         MOVE "UTL1" TO LABEL-ID.
         MOVE "THIS WAS PRODUCED BY BETA SECTION." TO LABEL-INFO.
         DISPLAY "521030 THIS LABEL CREATED:-" SAMPLE-LABEL.
         MOVE 1 TO LABEL-COUNT.
         GO TO MORE-LABELS.
     BETA-2.
         MOVE "UTL2" TO LABEL-ID.
         MOVE "THIS WAS THE SECOND LABEL PRODUCED BY BETA SECTION."
            TO LABEL-INFO.
         DISPLAY "521530 AND THIS LABEL TOO:-" SAMPLE-LABEL.
         MOVE 2 TO LABEL-COUNT.
         GO TO BETA-END.
     BETA-3.
         DISPLAY "522010 THIS LABEL READ:-" SAMPLE-LABEL.
         ADD 1 TO LABEL-COUNT.
         GO TO MORE-LABELS.
     BETA-END.EXIT.
     END DECLARATIVES.
     HERE-GOES.
         MOVE "0" TO I-O-INDICATOR.
```

Example 3-4 (format 1)

```
Data Division entries:

     FD  FILE-ONE.
           .
           .
           .
         LABEL RECORD IS ONE-LABEL.
           .
           .
           .
     01  ONE-LABEL.
         02 UHL-FIELD       PIC X(3).
         02 LABEL-NO        PIC 9.
         02 USER-INFO       PIC X(76).
     01  RECORD-ONE.
           .
           .
           .
     FD  FILE-TWO.
           .
           .
           .
         LABEL RECORD IS TWO-LABEL.
           .
           .
           .
     01  TWO-LABEL.
         02 UHL-FIELD       PIC X(3).
         02 LABEL-NO        PIC 9.
         02 USER-INFO       PIC X(76).
     01  RECORD-TWO.

Procedure Division statements:

     L SECTION.
         USE AFTER STANDARD LABEL PROCEDURE ON INPUT.
     L-1.
         IF UHL-FIELD NOT = "UHL" THEN STOP RUN.
            .
            .
            .
         GO TO MORE-LABELS.
     L-9.EXIT.
     M SECTION.
           .
           .
           .
     HOUSEKEEPING SECTION.
         OPEN INPUT FILE-ONE, FILE-TWO.

In this example, UHL-FIELD, LABEL-NO and USER-INFO are common label items.
Therefore, the unqualified reference to UHL-FIELD in the paragraph named
L-1 is legitimate.
```

Format 2

```
USE AFTER STANDARD {ERROR    } PROCEDURE
                   {EXCEPTION}

    ON {file-name-1 [file-name-2]...}
       {INPUT                       }
       {OUTPUT                      }
       {I-O                         }
       {EXTEND                      }
```

Rules for format 2

1. The ERROR and EXCEPTION phrases are equivalent and may be used optionally.
2. file-name must be defined in a file description (FD) entry of the program's Data Division.
3. When the file-name phrase is specified, the error handling procedures are executed only for the files named. No other USE procedures are performed for these files.
4. The specified procedures are executed by the system when an input/output error occurs during the execution of an input/output statement for a file.
5. Before execution of the user error routine, the standard system error routines for input/output error handling are executed.
6. After a USE procedure is executed, control passes to the calling routine.
7. INPUT indicates that the specified procedures are executed only for files opened as INPUT (OPEN statement with the INPUT phrase specified).
8. OUTPUT indicates that the specified procedures are executed only for files opened as OUTPUT (OPEN statement with the OUTPUT phrase specified).
9. I-O indicates that the specified procedures are executed only for files opened as I-O (OPEN statement with the I-O phrase specified).
10. EXTEND indicates that the specified procedures are executed only for files opened as EXTEND (OPEN statement with the EXTEND phrase specified).

Programming considerations for format 2

1. The USE procedures are performed:

   a) when an AT END condition occurs, provided that the input/output statement in which the AT END appears does not contain an AT END phrase.

   b) when a severe error occurs (FILE STATUS CODE ≥ 30).

   If there is no corresponding USE procedure for the file when a) or b) occurs, the program will abort.

2. Files referenced either implicitly (INPUT, OUTPUT, I-O and EXTEND) or explicitly (file-name-1, file-name-2, ...) in the USE statement need not have the same organization and access mode.

3. Within a USE procedure, there must be no reference to non-declarative procedures except for the PERFORM statement.

   Reference to procedure names which are subordinate to a USE statement may be made from another procedure, or non-declarative procedure, only by using a PERFORM statement.

4. file-name may not appear in more than one USE statement.

Example 3-5 (format 2)

Environment Division statements:

```
        SELECT MASTER-FILE ASSIGN DA-DISC-S-SYS001
               FILE STATUS INDICATOR.
    .
    .
    .
```

Data Division entries:

```
    FILE SECTION.
    FD  MASTER-FILE LABEL RECORD STANDARD.
    01  RECORD                  PIC X(80).
    WORKING-STORAGE SECTION.
    77  INDICATOR               PIC 99.
    77  CLOSE-INDICATOR         PIC X VALUE "0".
    .
    .
    .
```

Procedure Division statements:

```
    PROCEDURE DIVISION.
    DECLARATIVES.
    INPUT-ERROR SECTION.
        USE AFTER ERROR PROCEDURE ON MASTER-FILE.
    STATUS-QUERY.
        IF INDICATOR NOT < 30
            GO TO UNRECOVERABLE ERROR.
    END-OF-FILE.
        DISPLAY "END OF MASTER-FILE ENCOUNTERED".
        GO TO SPOL-BACK.
    UNRECOVERABLE-ERROR.
        DISPLAY "UNRECOVERABLE ERROR ("INDICATOR")"
            "FOR MASTER-FILE".
        IF CLOSE-INDICATOR = "1"
            CLOSE MASTER-FILE.
        DISPLAY "PROGRAM TERMINATED ABNORMALLY".
        STOP RUN.
    SPOOL-BACK.
        EXIT.
    END DECLARATIVES.
    START.
        OPEN INPUT MASTER-FILE.
        MOVE "1" TO CLOSE-INDICATOR.
        .
        .
        .
        MOVE "0" TO CLOSE-INDICATOR.
        CLOSE MASTER-FILE.
        .
        .
        .
```

WRITE STATEMENT

Function

The WRITE statement causes a data record to be released to an input/output or an output file. It can also be used for form feed in print files.

Format 1 of the WRITE statement must be used for sequential files.

Format 2 of the WRITE statement must be used for disk-storage files with any organization other than sequential (see chapters 4 and 5).

Format 1

```
WRITE record-name [FROM identifier-1]
    [ {AFTER  } [ {ADVANCING} ] { {identifier-2} [LINES] } ]
    [ {BEFORE } [ {           } ] { {integer     } [LINE ] } ]
                                   { {mnemonic-name}       }
                                   { {PAGE         }       }

    [AT {END-OF-PAGE} imperative-statement]
        {EOP        }
```

Rules

1. record-name and identifier-1 must not reference the same storage area.

2. record-name is the name of a data record in the File Section and must not be qualified by a file-name.

3. record-name must not be part of a sort-file.

4. identifier-2, if used in the ADVANCING phrase, must be the name of a numeric elementary integer data item.

   However, the compiler also permits use of nonnumeric elementary items for identifier-2.

5. The value of integer or the data item referenced by identifier-2 must be greater than or equal to zero or greater than 100.

   For files assigned to the symbolic device-name PRINTERANS, integer must be greater than 15 and less than 100.

   For files assigned to the symbolic device names PRINTER or PRINTERDOD, the values of integer or the data item referenced by identifier-2 are listed in Table 3-11.

6. If END-OF-PAGE is supplied, a LINAGE clause must be present as part of the file description entry for the corresponding file (see "LINAGE Clause").

7. The EOP and END-OF-PAGE phrases are equivalent.

8. mnemonic-name, if specified, must have an entry associated with it in the SPECIAL-NAMES paragraph.

9. mnemonic-name must not be supplied in the ADVANCING phrase if a LINAGE clause was specified for the file referenced in the WRITE statement.

10. The data record supplied by a WRITE statement is no longer available in the record area, unless the file related to the data record was specified in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful. The record is also available to any other files which may have been referenced in any SAME RECORD AREA clause together with the specified file.

11. Execution of a WRITE statement with the FROM phrase is equivalent to the following statements:

    ```
    MOVE identifier-1 TO record-name.
    WRITE record-name.
    ```

    Data is transferred according to the rules for a MOVE statement without the CORRESPONDING phrase.

    The contents of the record area before execution of the implicit MOVE statement have no effect on the execution of the WRITE statement.

    After the WRITE statement is successfully executed, the information is still available in the area referred to by the identifier; however, as pointed out under Rule 11, this is not necessarily applicable to the data record area.

12. If the end-of-volume condition is encountered on a multivolume output file for tape or disk storage and sequential access mode, then execution of the WRITE statement causes the following steps to be performed:

    a) The standard volume header label procedures and the user volume trailer label procedures specified via a USE procedure are performed. The order in which these procedures are executed depends on the BEFORE/AFTER phrases supplied in the USE procedure, if any.

    b) A volume switch takes place.

    c) The standard volume header label procedures and the user volume header label procedures specified via a USE procedure are performed. The order in which these procedures are executed depends on the BEFORE/AFTER phrases supplied in the USE procedure, if any.

13. After execution of a WRITE statement, the value of the data item of any FILE STATUS clause existing for that file is updated (see also "FILE STATUS Clause").

14. The number of character positions in the data record indicated by record-name cannot exceed the largest record description entry of the associated file, nor can it be smaller than the number of character positions permitted by the associated RECORD clause. If one of these cases occurs, the WRITE statement will terminate abnormally, and no writing will take place. The contents of the record area will remain unchanged, and the I-O status of the file associated with record-name will be set to a value indicating the cause of the condition (see "I-O Status").

15. Both the ADVANCING and the END-OF-PAGE phrases provide a capability for vertically positioning every single line within a page to be printed.

16. Use of the BEFORE/AFTER ADVANCING phrase causes the data record to be printed either before or after the page to be printed is advanced according to the rules described below.

17. If the symbolic device name PRINTERANS was used in the ASSIGN clause of the file, the ADVANCING phrase causes the page to be printed to be advanced according to the following rules:

    a) If identifier-2 (elementary numeric data item) is present, the page is advanced by the number of lines defined by the value of identifier-2.

    b) If integer is present, the page is advanced by the number of lines defined by the integer.

    c) If PAGE is present, the data record is printed either before or after positioning to the beginning of the first line on a new page. The beginning of a new page is defined either by the physical properties of the printer (advance to channel 1) or, if a LINAGE clause is present, by the beginning of the next logical page (on this, see also "LINAGE Clause").

18. If the symbolic device name SYSLST, PRINTER or PRINTERDOD was used in the ASSIGN clause of the file, additionally indicating that no LINAGE clause must be defined in the file description entry, use of the ADVANCING phrase causes the page to be printed to be advanced according to the following rules:

    a) If identifier-2 (nonnumeric or numeric elementary integer data item), integer, or mnemonic-name are present, the page is advanced according to the rules shown in table 6-23.

    b) If PAGE is specified, the data record is printed either before or after positioning to the beginning of the first line of a new page. In this case, the beginning of a new page is determined by the physical properties of the printer (advance in channel 1).

19. If the ADVANCING phrase is omitted, and the symbolic device name PRINTERANS is supplied in the ASSIGN clause, the printer will advance automatically, corresponding to the AFTER ADVANCING 1 LINE phrase. The ADVANCING phrase is considered to override the automatic advance function.

    For reasons of compatibility, the compilers discussed in this manual provide spacing according to the BEFORE ADVANCING 1 LINE phrase, if the ADVANCING phrase is omitted and PRINTER was specified in the ASSIGN clause. If PRINTERDOD was specified in the ASSIGN clause, the carriage control character supplied by the user in the first character of the data record is used for advance control

    See also Table 3-12 for further details.

20. If the end of a logical page is encountered during execution of a WRITE statement with the END-OF-PAGE phrase specified, control is transferred to the imperative statement which follows the AT END-OF-PAGE phrase. The logical end of a page is defined by the LINAGE clause in the file description entry of that file.

21. An END-OF-PAGE condition occurs when execution of a WRITE statement within the END-OF-PAGE phrase specified causes printing or spacing within the page footing area. This is the case whenever, during the execution of such WRITE statement, the value of the special register LINAGE-COUNTER becomes equal to or greater than the value of integer-2 of the LINAGE clause date item referenced by identifier-2. In this case, the WRITE statement is executed, and control is then transferred to the imperative statement which follows the END-OF-PAGE phrase.

22. An automatic page-overflow condition occurs whenever a given WRITE statement (whether or not the AT END-OF-PAGE phrase us specified) can no longer be executed within the current page limits.

    This is the case whenever, during the execution of such WRITE statement, the value of the special register LINAGE-COUNTER becomes equal to or greater than the value of integer-1 or the LINAGE clause data item referenced by integer-1. In this case, the data record is printed either before or after positioning to the first line of the next logical page (depending on the BEFORE/AFTER phrase). This line is defined by specification of the LINAGE clause. The imperative statement specified in the END-OF-PAGE phrase is executed after the data record was written and the device was repositioned.

    If the integer-2 or data-name-2 phrases of the LINAGE clause are not used, the END-OF-PAGE condition will occur whenever a WRITE statement is executed with EOP phrase specified and the value in the special LINAGE-COUNTER register is greater than or equal to the value of integer-1 or the LINAGE clause data item referenced by identifier-1.

    If execution of a WRITE statement produces a value of the special register LINAGE-COUNTER which is equal to or greater than the value of integer-2 or the data item referenced by data-name-2 as well as being equal to or greater than the value of integer-1 or the data item referenced by data-name-1, then the statement will be executed as if integer-2 or data-name-2 had not been specified, even when the integer-2 or data-name-2 is present in the LINAGE clause.

23. When the ADVANCING phrase is used together with the LINAGE clause for a device which does not represent a physical printer, then any blank lines lines at the top and bottom margins as well as within the page body are represented, on the appropriate storage medium, as data records containing blank characters.

24. Execution of a WRITE statement which attempts to write outside the physical boundaries of a sequential file results in an exception condition; and the following steps are performed:

    a) The value of the FILE STATUS data item of the file is set to indicate violation of the boundaries of the file (see "FILE STATUS Clause").

    b) Any USE ERROR/EXCEPTION procedure that may be explicitly or implicitly supplied for this file will be executed.

    c) If there is neither an explicit nor an implicit USE procedure for this file, the program will terminate abnormally.

25. Execution of a WRITE statement means that the same as execution of a REWRITE statement if the following conditions are satisfied:

    a) The file was opened I-O (OPEN statement with I-O phrase).

    b) It is a sequential file whose access mode is sequential.

Programming considerations

1. When a WRITE statement is executed, the file must have been opened using the OUTPUT, I-O, or EXTEND phrases in the OPEN statement.

2. Table 3-10 gives a detailed account of the use of the first character in the data record, depending on the symbolic device names of the ASSIGN clause and the WRITE statement phrases.

3. When a WRITE statement for a file with the symbolic device name PRINTER includes the ADVANCING phrase, then all of the WRITE statements referring to that file must include the ADVANCING phrase.

   In all other cases (symbolic device name PRINTERANS or PRINTERDOD), mixed use of WRITE statements with or without ADVANCING phrases is legitimate (see also Table 3-10).

4. When a WRITE statement is used with AFTER-ADVANCING phrase, the printer is positioned to the line requested and is advanced one additional line after printing.

   In the case of PRINTERANS, a WRITE AFTER phrase is performed as follows:

   a) An empty record is printed with corresponding form-feed specification (WRITE BEFORE).

   b) The record is written.

   For all devices, form-feed suppression is only possible when the WRITE BEFORE phrase is specified, except in the case of PRINTERANS, where the BEFORE phrase reduces the number of records to be output.

EXAMPLE 3-6

Environment Division entry:

```
          .
          .
          .
       SELECT PRINT ASSIGN TO UR-PRINTER-S-SYS020.
          .
          .
          .
```

Data Division entries:

```
          .
          .
          .
    FD  PRINT
    01  PRINTER-RECORD.
        02 CONTROL-CHARACTER  PIC X.
        02 PRINTER-LINE       PIC X(132).
          .
          .
          .
    77  NUMBER-1             PIC 99.
          .
          .
          .
```

Procedure Division statements:

```
          .
          .
          .
       MOVE 1 TO NUMBER-1.
          .
          .
          .
       WRITE PRINTER-RECORD BEFORE ADVANCING NUMBER-1 LINES.
          .
          .
          .
       WRITE PRINTER-RECORD AFTER ADVANCING 0 LINES.
```

In this example, the first character of the data record was reserved for the carriage control character (in accordance with the rules as shown in Table 3-12 for the symbolic device name PRINTER). The first WRITE statement write PRINTER-RECORD; then the printer page is advanced one line (contents of NUMBER-1 = 1). The second WRITE statement positions at the top of a new page (special significance of 0 lines for PRINTER); then the current contents of PRINTER-RECORD are written.

<table>
<tr><th>Indicator</th><th>Format</th><th colspan="2">Action</th></tr>
<tr><td rowspan="3">identifier-2</td><td>Represents a non-negative integer item with a value less than 16.</td><td colspan="2">Printer page is advanced n lines, where n is the current value of the item represented by identifier-2.</td></tr>
<tr><td rowspan="2">Represents a 1-character item (PICTURE X).</td><td>Value</td><td>Meaning</td></tr>
<tr><td>blank
0
-
+
1-8

A,B</td><td>Single-spacing
Double-spacing
Triple-spacing
Suppress spacing
Skip to channel 1-8, respectively
Skip to channel 10 and 11 respectively.</td></tr>
<tr><td>integer</td><td>A nonnegative integer less than 16.</td><td colspan="2">If integer = 0, the paper is ejected. Otherwise, the printer page is advanced n lines, where N = integer.</td></tr>
<tr><td rowspan="2">mnemonic-name</td><td rowspan="2">A name related to an implementor-name in a SPECIAL-NAMES paragraph.</td><td>implementor-name</td><td>Action</td></tr>
<tr><td>CSP
C01 to C08
C10 OR C11</td><td>Suppress spacing
Skip to chan. 1-8
Skip to chan. 10-11</td></tr>
</table>

Table 3-9 Format and meaning of spacing indicators in a WRITE ADVANCING statement for PRINTER and PRINTERDOD

Device-name	WRITE statement without ADVANCING phrase		WRITE statement with ADVANCING phrase	Comments
SYSLST	Standard spacing when ADVANCING omitted is the same as if AFTER 1 LINE is specified; the first character of the data record is available for user data and is included in the print.	BS1000: Space control character is not generated. Printer is automatically positioned to the next line. BS2000: The user must make sure that the record length he defines is 1 byte smaller than the maximum length supported by the device: the compiler automatically generated 1 byte at the front of the record.	The user must reserve the first character a logical data record; in this place, the runtime system enters the carriage control character at object time. Any user data contained is overwritten.	Mixed use of WRITE statements both with and without specification of the ADVANCING phrase for the same file within one program is not permitted. If this case does occur, and ADVANCING is omitted, the first character of the data record will be interpreted as a control character; therefore it must contain a valid control character.
PRINTER	The first character in the data record is interpreted as a space control character: an implicit WRITE ... AFTER ADVANCING 1 LINE is performed.			
PRINTERDOD	Spacing is controlled by the first character in each logical data record; the user, therefore, must supply there the appropriate control character before every execution of such WRITE statement.			Mixed use of WRITE statements both with and without specifications of the ADVANCING phrase is permitted. In either case, however, the user information of the printer record begins only with the second character of the data record.
PRINTERANS	Standard spacing when ADVANCING is omitted as if AFTER 1 LINE is specified; the first character of the data record is available for user data.		The first character of the data record is available for user data.	The place for the carriage control character is reserved by the compiler and is not accessible to the user. This type of printer supports specification of the LINAGE clause in the file description entry. Write statements both with and without the ADVANCING phrase specified are allowed for a given file.

Table 3-10 Use of PRINTER device-name in connection with the WRITE statement

From

Name . . . . . .

Company . . . . . .

Street . . . . . .

City/Postal Code . . . . . .

Phone . . . . . .

Date . . . . . .

To
Siemens AG

K D ST QM 2
Manualredaktion

Otto-Hahn-Ring 6
D-8000 München 83

Reader's Reply Form

COB1 (BS2000)
COBOL Compiler

Reference Manual Part 1
Edition August 1986 (Software Product COB1 V2.3A)

Order No. U343-J-Z55-6-7600

Page	Criticisms/Suggestions/Corrections

Page	Criticisms/Suggestions/Corrections

From

Name . . . . . .

Company . . . . . .

Street . . . . . .

City/Postal Code . . . . . .

Phone . . . . . .

Date . . . . . .

To
Siemens AG

K D ST QM 2
Manualredaktion

Otto-Hahn-Ring 6
D-8000 München 83

Reader's Reply Form

COB1 (BS2000)
COBOL Compiler

Reference Manual Part 1
Edition August 1986 (Software Product COB1 V2.3A)

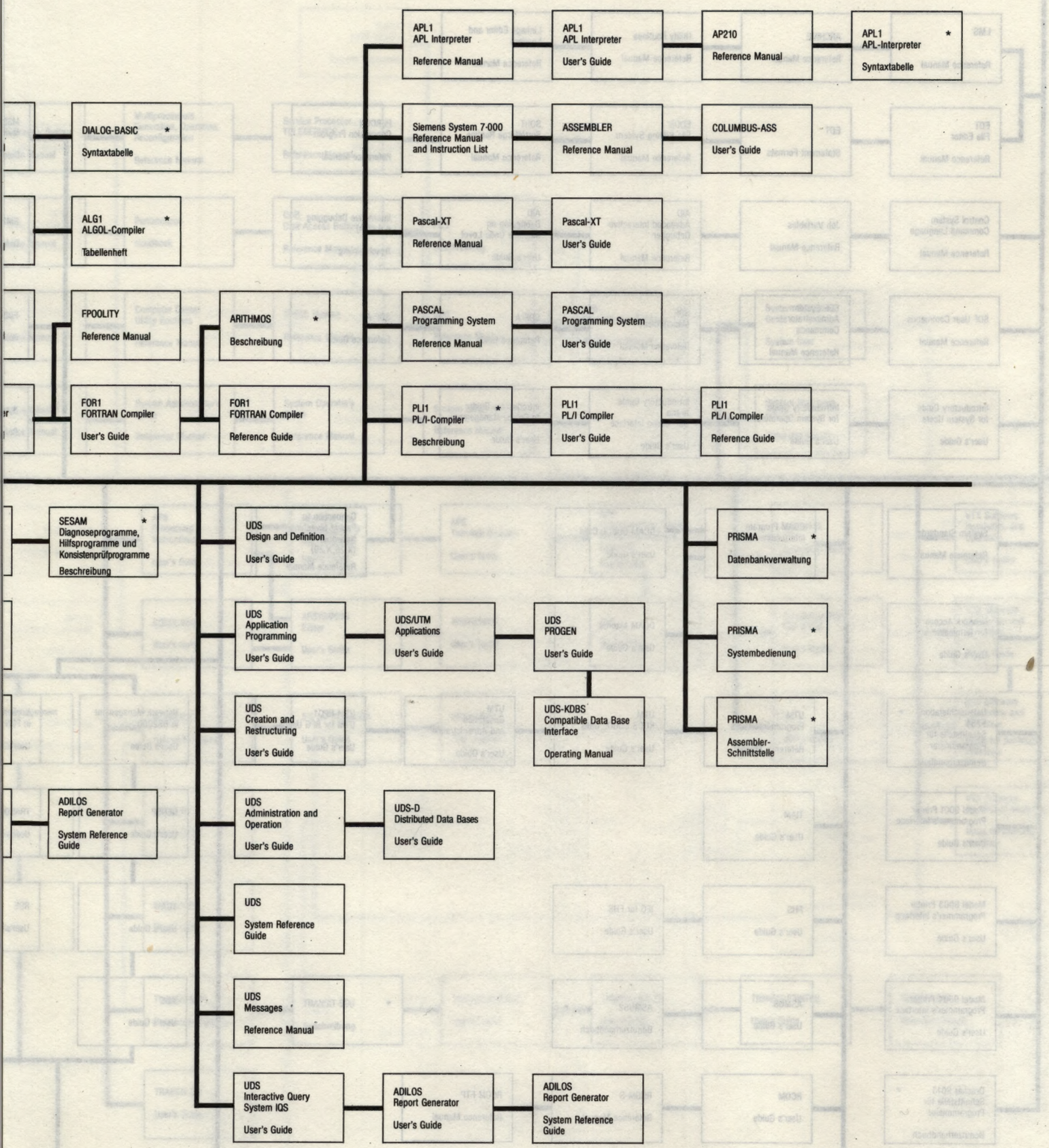
Order No. U343-J-Z55-6-7600

Page	Criticisms/Suggestions/Corrections

Page	Criticisms/Suggestions/Corrections

Published by Bereich Datentechnik
Postfach 83 09 51, D-8000 München 83

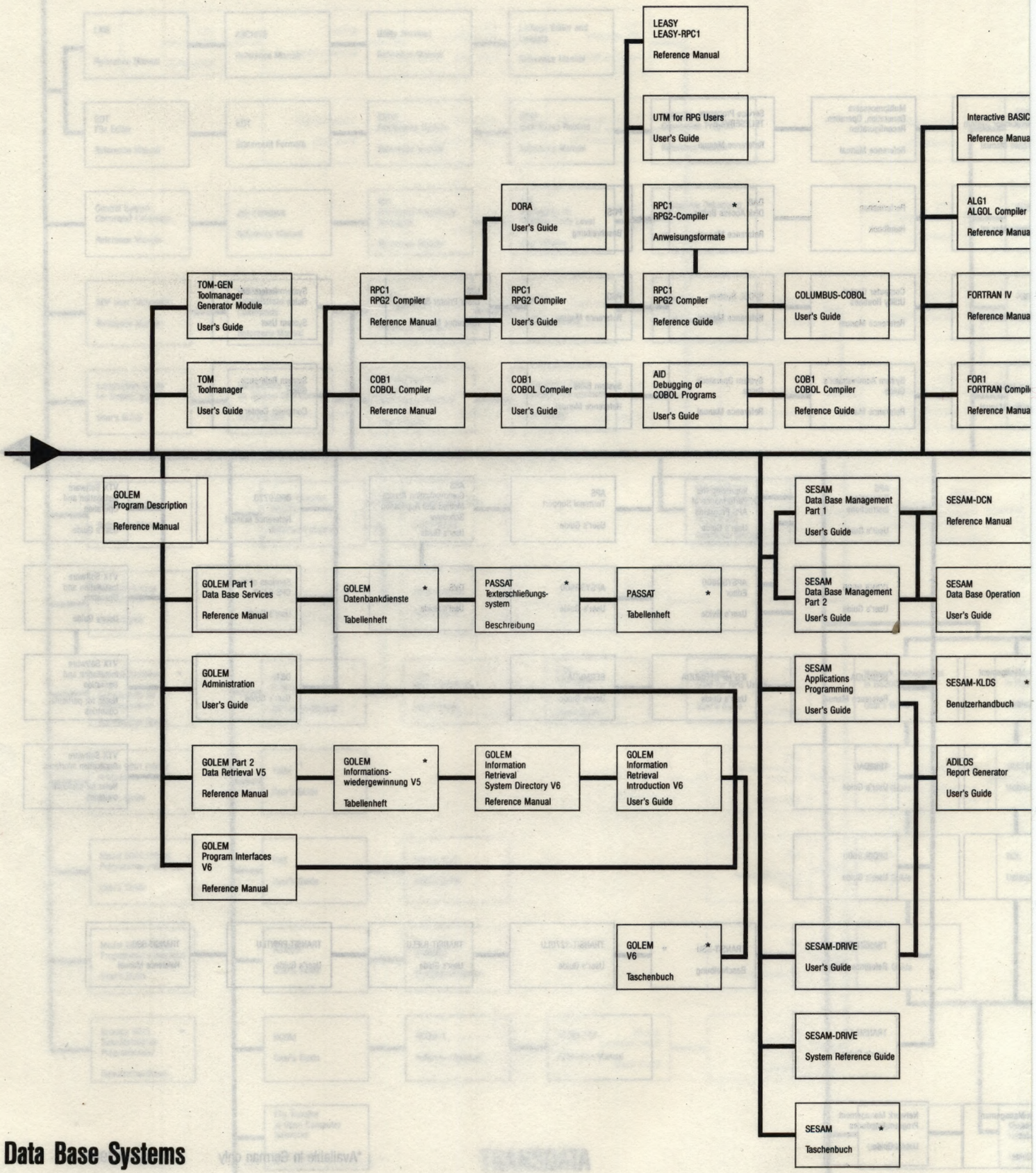
Siemens Aktiengesellschaft



* Available in German only.

August 1986

Programming Systems



Data Base Systems

BS2000

System Messages
Reference Manual

Systems Standards
Reference Manual

DMS
Tape Processing
Reference Manual

DMS
Disk Processing
Reference Manual

Executive Macros
Reference Manual

LEASY
Reference Manual

RFA
Reference Manual

ADAM
Reference Manual

LMS
Reference Manual

ARCHIVE
Reference Manual

Utility Routines
Reference Manual

Linkage Editor and
Loaders
Reference Manual

EDT
File Editor
Reference Manual

EDT
Statement Formats

EDOR
File Editing System
Reference Manual

SORT
Sort/Merge Routine
Reference Manual

PERCON
Conversion Program
Reference Manual

Control System
Command Language
Reference Manual

Job Variables
Reference Manual

AID
Advanced Interactive
Debugger
Reference Manual

AID
Debugging on
Machine Code Level
User's Guide

Interactive Debugging
Aid (IDA)
Beschreibung

SDF User Commands
Reference Manual

SDF System
Administrator
Commands
Reference Manual

SDF
Management
Reference Manual

SDF-A
Reference Guide

SDF-A
Reference Guide

Introductory Guide
for System Users
User's Guide

Introductory Guide
for System Operators
User's Guide

Introductory Guide
to the
SDF Dialog Interface
User's Guide

Introductory Guide
to Device Management
User's Guide

System Standards
Reference Manual

Network Access
for Terminals
User's Guide

Datensichtstation 9754 *
Schnittstelle für
Programmierer
Benutzerhandbuch

Model 9001 Printer
Programmer's Interface
User's Guide

Model 9003 Printer
Programmer's Interface
User's Guide

Model 9004 Printer
Programmer's Interface
User's Guide

Drucker 9013 *
Schnittstelle für
Programmierer
Benutzerhandbuch

DCAM Program
Interfaces
Reference Manual

DCAM COBOL Calls
User's Guide

DCAM Macros
User's Guide

UTM
Program Interfaces
Reference Manual

UTM
KDCS COBOL Calls
User's Guide

UTM
Generation
and Administration
User's Guide

UTM-RPC1
UTM for RPG Users
User's Guide

TIAM
User's Guide

FHS
User's Guide

IFG for FHS
User's Guide

ASMUS
User's Guide

ASMUS2 *
Benutzerhandbuch

RCOM
User's Guide

RCOM-S
Reference Manual

RCOM FTP
Reference Manual

File Transfer
in Open Computer
Networks
User's Guide

Teletex-Anschluß *
an BS2000 mit TTX-B
Benutzerhandbuch

Connection to
Packet Switching
Networks
(X.25, X.29)
Reference Manual

Network Management
in BS2000
User's Guide

NEOSP
User's Guide

TDA
User's Guide

ADS
User's Guide

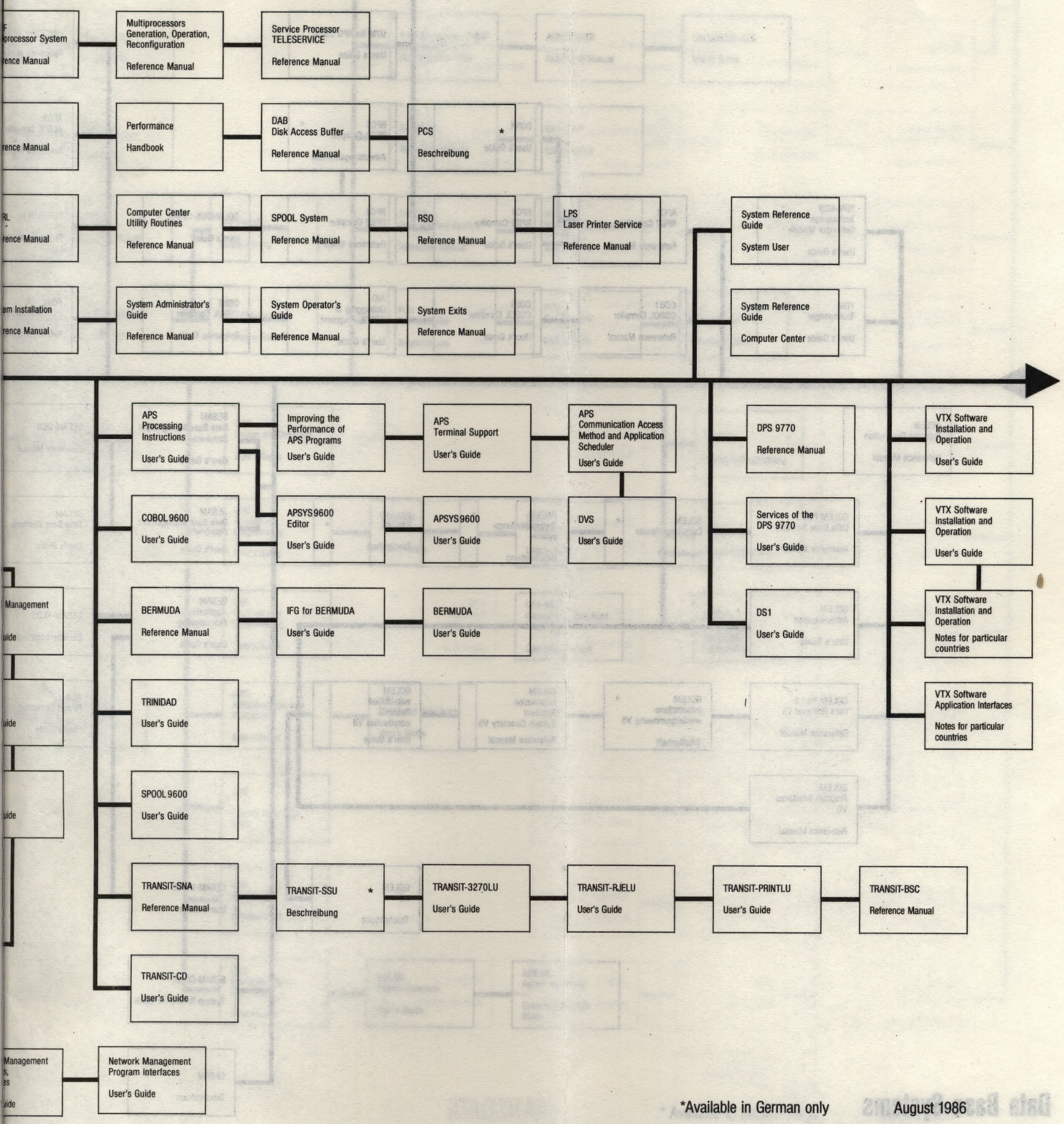
RBAS
User's Guide

OMNIS
User's Guide

TRANSDATA

Publications Plan for

- BS2000
- TRANSDATA
- Programming Systems
- Data Bases



*Available in German only

August 1986